The Report Committee for Jonathan Barten Stanley

certifies that this is the approved version of the following report:

Tradeoffs in Parallel Prefix Adder Structures

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____

Earl E. Swartzlander, Jr.

_____

Lizy K. John

Tradeoffs in Parallel Prefix Adder Structures

by

Jonathan Barten Stanley, B.S.E.E.

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirement

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2015

Acknowledgements

This work would not have been possible without acknowledging the following people:

My parents, John L. and Joan B. Stanley. They instilled in me the value of lifelong learning and fostered inquisitiveness in all things life.

All the teachers, friends, and fellow peers who have provided support and encouragement.

Dr. Earl Swartzlander, who graciously shared his knowledge and experience in the field of computer arithmetic, as well as for his recommendation to take my class project further to create this report.

Tradeoffs in Parallel Prefix Adder Structures

by

Jonathan Barten Stanley, M.S.E.

The University of Texas at Austin, 2015

SUPERVISOR: Earl E. Swartzlander, Jr.

This report presents the results of research on comparing the structures and qualities of fast parallel prefix adders. The binary adder serves as a fundamental component of many digital arithmetic operations. Many modern microprocessors and ASICs that require high speed arithmetic logic often implement parallel prefix adders. Modern parallel prefix adder structures are based on previous works including those of Kogge-Stone, Brent-Kung, Ladner-Fischer, Knowles, *et al*. and designs presented in each work have their own merits and tradeoffs that are suitable for certain applications.

Previous works have described standard and systematic ways to design and construct functional parallel prefix adder structures. Although the parallel prefix adder has been studied for decades, this work explores the possibility that non-standard and more optimal structures may exist by developing and utilizing a brute force search algorithm based on the prefix operator rules and properties to find all possible parallel prefix adder structures. The parallel prefix adder search algorithm design, search results and study of tradeoffs are discussed in this work.

Table of Contents

1 Problem Definition

Parallel prefix adder structure designs are heuristic-based and are not described by a formally defined set of equations with generalized and useful design variables. Previous works have described standard methods for constructing a parallel prefix adder structure that optimize modular layout (Brent-Kung [1]), fanout (Kogge-Stone [2]), and complexity (Ladner-Fischer [3]). Hybrid parallel prefix adders have been described in several other works over the years including Knowles [4] and Han-Carlson [5].

This work aims to answer the question of whether there exists a prefix adder structure that may not implement a systematic structure and exhibits an improvement in key design areas (delay, complexity, fanout, area and power) compared to known standard structures.

2 Introduction

Parallel prefix adders are based on the fundamental concept behind carry lookahead adder designs. For two given inputs A and B with N-bit vector representations $a_{N-1}, \ldots, a_0$ and $b_{N-1}, \ldots, b_0$ respectively, the carry behavior for each bit column i can be predicted by producing generate $g_i = a_i b_i$ and propagate $p_i = a_i \oplus b_i$ signals in the pre-processing stage. When summing together bits a_i and b_i, the generate signal indicates that a carry will be generated regardless of a carry input, and the propagate signal indicates that the carry input will propagate to the carry output.

The prefix computation stage is the heart of the prefix adder and is responsible for producing group generate $g_{i:0} = g_i + p_i g_{i-1:0}$ and group propagate $p_{i:0} = p_i p_{i-1} \ldots p_1 p_0$ signals for each bit column i. When summing together bit vectors $a_i, \ldots, a_0$ and $b_i, \ldots, b_0$, the group generate signal indicates that a carry output will be generated regardless of the carry input, and the group propagate signal will propagate the carry input to the carry output. The prefix operator is the fundamental operation of the prefix computation stage. The operator allows group generate and propagate signals for bit columns i:0 to be created from a subset of group signals i:k and k-1:0. Two properties of the prefix operator are associativity, which allows k to be any value between 0 and i, and idempotency, which allows overlap or redundancy around the boundary k. The prefix operator grouping logic is usually illustrated as a single node in a parallel prefix adder design structure diagram.

In the post-processing stage of a prefix adder, the sum output S represented by a bit vector $s_{i-1}, \ldots, s_0$ is produced by $s_i = p_i \oplus c_i$ signals where $c_i = g_{i-1:0} + p_{i-1:0} c_{in}$. As a result, the critical timing path of a parallel prefix adder is essentially governed by a series of 2-input logic combinations of generate and propagate signals that can be arranged efficiently but the arrangement is currently a heuristic approach. An efficient implementation achieves a delay on the order of $\log_2 N$ for N bits and allows the prefix adder to realize a significant improvement in speed relative to a ripple carry adder whose delay grows linearly as the adder size grows.

Parallel prefix adder structures have been studied for a long time because of their ubiquity in many digital systems and their value in high speed digital arithmetic. Another advantage of the prefix structure is its capability to be pipelined for higher throughput. The following sections provide an overview of previous key works on parallel prefix adders with a description of their design characteristics and tradeoffs.

2.1 Kogge-Stone

The Kogge-Stone type of adder is based on a set of mathematical rules described by Kogge and Stone in [2]. Figure 1 shows a 16-bit Kogge-Stone parallel prefix adder with minimum depth of 4 and 49 nodes. The high complexity can be readily discerned in Figure 1 by the amount of nodes and wires. The design comes with the benefit that all nodes have a maximum fanout of 1 and is the fastest design when neglecting wire capacitance. Fanout here is defined as the number of branches to nodes located in other bit columns and does not count the implicit branch to the next row in the same bit column.

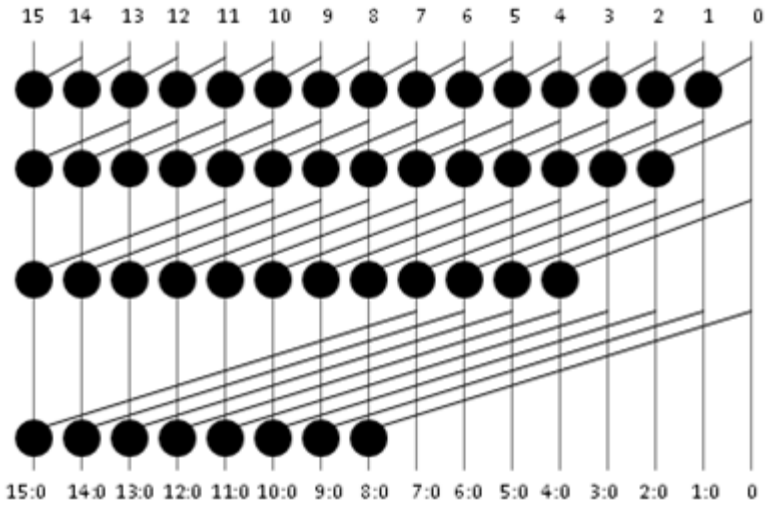


Figure 1 – Kogge-Stone 16-bit, 4-level, 49-node adder

2.2 Ladner-Fischer

Ladner and Fischer applied the concept of binary recursion to the parallel prefix problem in [3]. Their paper describes a set of rules for designing a family of adders that trades off between depth and complexity by varying the fanout.

Figure 2 is best described as a Ladner-Fischer 16-bit adder with minimum depth of 4 and a maximum 32 nodes. The consequence of this design is the high fanout of 8 in the final level, which introduces more gate capacitance and delay on the node circuit driving the fanout.

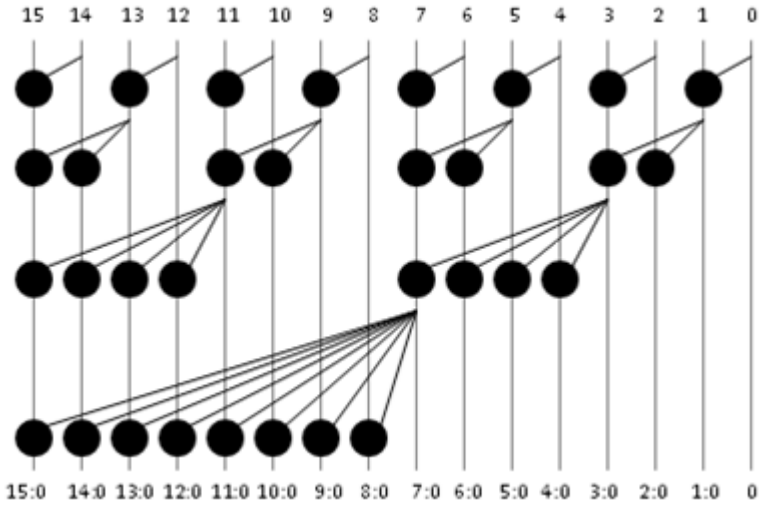


Figure 2 – Ladner-Fischer 16-bit, 4-level, 32-node adder

Figure 3 is an example of a Ladner-Fisher 16-bit adder variant that reduces complexity and fanout with node count of 27 by increasing the depth to 5 from the minimum depth of 4. Fanout is also reduced to 4 from the maximum fanout of 8. It exhibits characteristics in between the minimum depth Ladner-Fischer version and a Brent-Kung adder described in the next section.

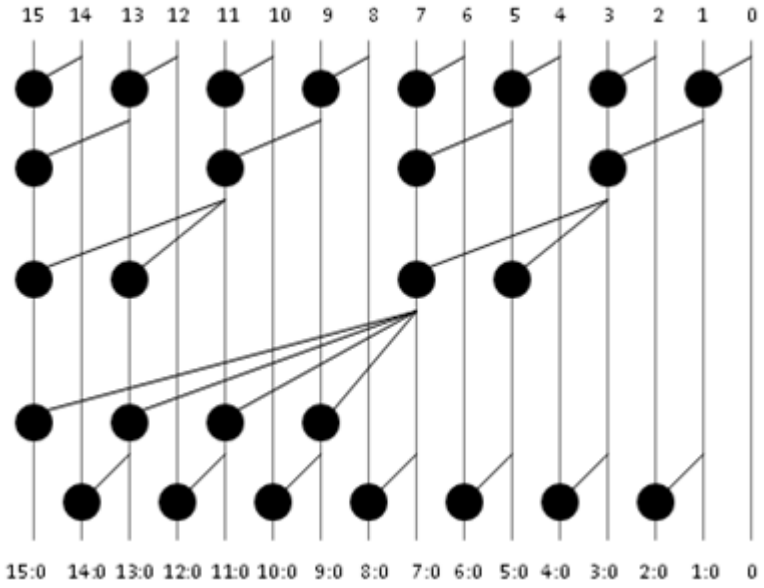


Figure 3 – Ladner-Fischer 16-bit 5-level 27-node adder

2.3 Brent-Kung

The parallel prefix adder illustrated in Figure 4 was described by Brent and Kung in [4]. They were primarily concerned with the VLSI challenges at the time and cost of high complexity and fanout of the Kogge-Stone and Ladner-Fischer designs. The Brent-Kung parallel prefix adder design was based on the following restrictions: fanout = 1, at most 2 wire crossings, and all nodes are implemented uniformly with identical gate sizes. The penalty of minimizing complexity is an increase in depth to $2 \log_2 N - 1$ for an N-bit adder. For this reason, the Brent-Kung adder is considered to be a minimal complexity design and generally slower due to the depth.

The 16-bit Brent-Kung adder in Figure 4 actually has a slight deviation from the true design described in the original paper because there is a fanout of 2 in the 4^{th} level on the 8^{th} bit column to reduce the depth to 6. However, this variation is still generally considered a Brent-Kung parallel prefix adder structure.

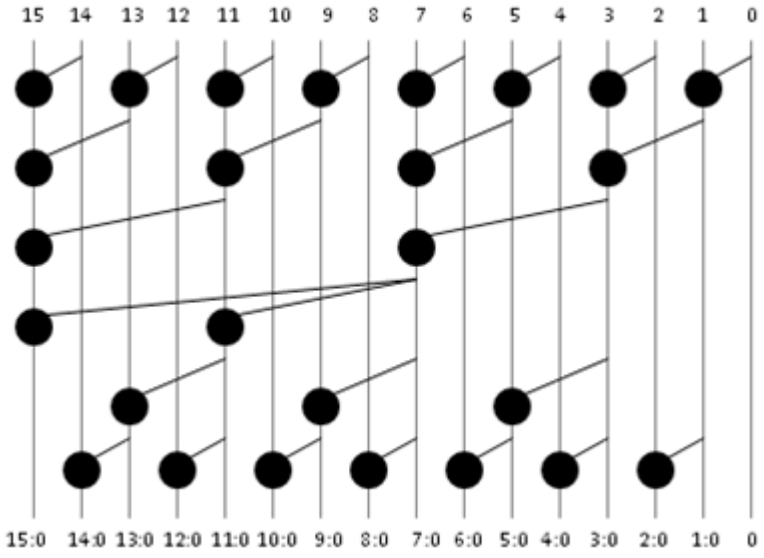


Figure 4 – Brent-Kung 16-bit 6-level 26-node adder

2.4 Han-Carlson

Han and Carlson described an approach for designing parallel prefix adders using design characteristics from Kogge-Stone and Brent-Kung [5]. The tradeoff is an increase in number of nodes for a decrease in depth, so a Han-Carlson adder is generally expected to perform faster than Brent-Kung without the heavy hardware cost of a Kogge-Stone. Figure 5 is a Han-Carlson 16-bit adder with the first and last rows resembling Brent-Kung adder and the inner rows resembling a Kogge-Stone adder. It has a depth of 5 levels and has the same number of nodes (32) as the minimum depth Ladner-Fischer.

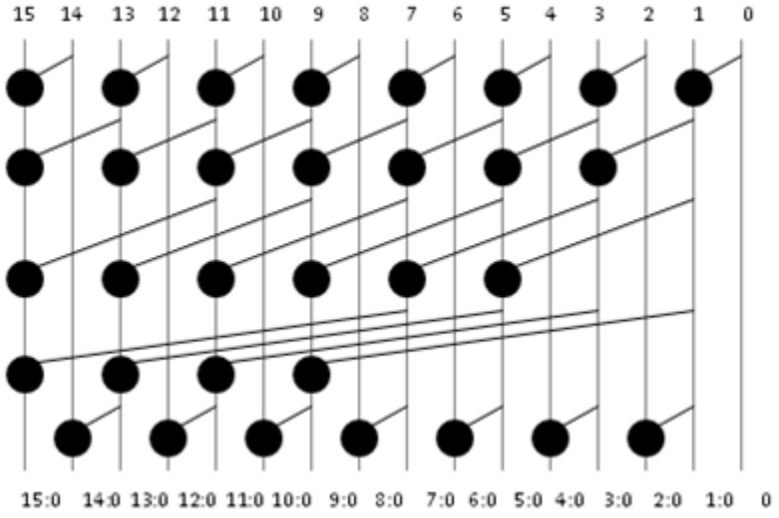


Figure 5 – Han-Carlson 16-bit 6-level 32-node adder

2.5 Knowles

The Knowles parallel prefix adder designs trade off between fanout and complexity while maintaining the minimum depth [4]. Thus, the Knowles class of adders is a hybrid with characteristics that lie between Kogge-Stone and Ladner-Fischer. Figure 6 is a Knowles (1,2,2,4) 16-bit adder in which the max fanout in levels 1, 2, 3, and 4 are 1, 2, 2, and 4 respectively. There are several other Knowles adders within the family with different maximum fanouts in each level that are not shown here.

A characteristic of Knowles adders is that it allows idempotency and thus has some redundancy. In Figure 6 for instance, the final node on bit 8 groups together inputs 8:2 and 3:0 with redundancy on bits 3:2 to generate the group signals 8:0.

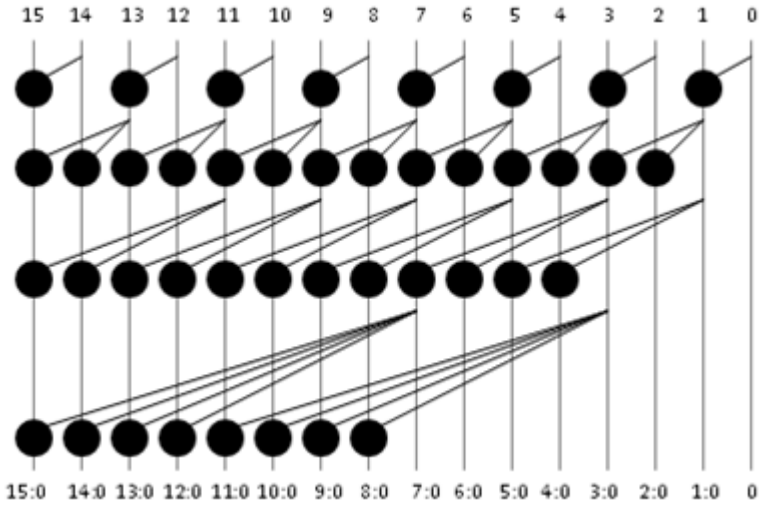


Figure 6 – Knowles (1,2,2,4) 16-bit 4-level 42-node adder

2.6 Summary of Prefix Adders

Table 1 shows a summary of existing parallel prefix adder designs and their key design variables. The most optimal parallel prefix adder structure is one that is able to minimize all three design variables and resides closer to the origin in the 3D plot of Figure 7. This report will show that parallel prefix adders with a non-standard structure and more optimal characteristics do exist.

Table 1 – Summary of Parallel Prefix Adder Characteristics

Structure	Depth	Max Fanout	Nodes
Kogge-Stone [2]	$\log_2 N$	1	$N \log_2 N - N + 1$
Ladner-Fischer [3]	$\log_2 N$	N/2	$0.5N \log_2 N$
Brent-Kung [1]	$2 \log_2 N - 1$	1	$2N - 2 - \log_2 N$
Han-Carlson [5]	$\log_2 N + 1$	1	$0.5N \log_2 N$
Knowles (1,…,F) [4]	$\log_2 N$	F	Varies

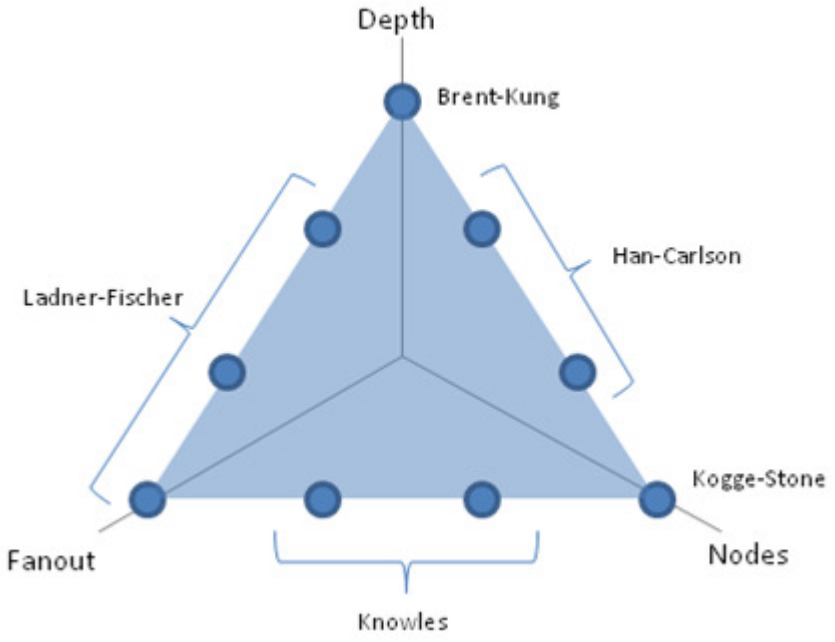


Figure 7 – 3D characteristic plot of existing parallel prefix adders

3 Study Approach

A search algorithm was developed to execute a relatively efficient brute force search for all possible valid parallel prefix adder structures for a given value of N bits. The first step in designing the search algorithm requires a method for describing a prefix adder structure. Choi and Swartzlander have described a method for representing a complete set of Knowles prefix adders with matrices in [6]. The matrix representation was used as a standard way to produce a set of Spice simulation results for characterizing delay and transistor widths.

However, this work aims to create a search algorithm that discovers all possible radix-2 prefix adder structures and then examine the tradeoffs between number of nodes (complexity), fanout, and number of levels (delay). The expectation is that the brute force search could reveal non-standard and other novel hybrid prefix adder structures. The requirement for the matrix representation is simpler by only showing the location of identically sized nodes with fixed radix of 2, which is different than the matrices described in [6].

3.1 Prefix Structure Matrix Representation

Concept: For row (i.e., level) j+1, the i^{th} column value represents the column index value of the signal source in the preceding row j. If no prefix computation exists, that is, a buffer is used instead, then the i^{th} column value equals i.

Figure 8 illustrates the proposed matrix representation for 8-bit prefix adder structures to be used by the search algorithm. The bit columns are ordered from right to left for i = 0 to N-1 to be consistent with the dot diagram representation of the prefix adder structures above. Note that in the Brent-Kung table example assumes a fanout of 2 for column 4 (i=3) in level 3 rather than following the strict definition described in the Brent-Kung paper [4].

Kogge-Stone

6	5	4	3	2	1	0	0
5	4	3	2	1	0	1	0
3	2	1	0	3	2	1	0

Ladner-Fischer

6	6	4	4	2	2	0	0
5	5	5	4	1	1	1	0
3	3	3	3	3	2	1	0

Brent-Kung

6	6	4	4	2	2	0	0
5	6	5	4	1	2	1	0
3	6	3	4	3	2	1	0
7	5	5	3	3	1	1	0

Figure 8 – Matrix representation of 8-bit parallel prefix adders

3.2 Prefix Structure Search Algorithm

The search algorithm is provided with two input parameters N-bits and M-rows to describe the size of the prefix computation stage. A basic naïve brute-force search would generate every possible pattern of a N×M matrix with each entry having N possible values, which would yield N^{NM} possible combinations. A small N=8 bit adder with M=4 rows would yield $8x10^{28}$ possible combinations, which is a highly inefficient use of computing time and a single desktop computer will be incapable of completing the search within reasonable time.

There are several important observations of the prefix structure that can be made to optimize the search algorithm:

1. For the i^{th} bit column, group signal sources are always from the i^{th} column or less. That is, group i:0 will never receive signals from i+1 or higher. So the maximum values in each matrix cell are reduced as shown in Figure 9.

7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7

→

7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0

Figure 9 – Adjustment of Max Values in Matrix

The key improvement of this realization is this reduces the number of possible patterns from N^{NM} to $(N!)^M$

2. The first 2 bit columns (i=0 and i=1) are always the same, so the algorithm does not need to try different combinations in those columns because there is only one valid column pattern for all prefix adder types. This provides a modest reduction to the number of possible patterns from $(N!)^M$ to $((N-2)!)^M$
3. Valid structures in row j+1 will always contain valid structures in all previous rows j or less. In other words, it is more efficient to test and gather all valid structures for the first row 1, and then test and gather all valid structures in the next row 2 using the subset of valid structures from row 1, and so on. There is no prediction on how many possible valid structures exist per row in order to estimate the algorithm speedup, but it will provide significant improvement in the efficiency of the search.
4. Idempotency can be disallowed to further reduce the number of possible patterns.

For a N=8 bit adder with M=4 rows, the worst-case number of possible patterns is $((N-2)!)^M = 2.7 \times 10^{11}$ assuming no speedup benefit from observation 3 above. If a single desktop computer can compute about 100,000 patterns per second then it would take an estimated 31 days to discover and exhaust all possible patterns. Ideally, the algorithm would be able to complete within reasonable time for larger N-bit adders, but the number of possible prefix structures increases exponentially with N and search time increases accordingly. For the purpose of this work, the search algorithm will find all possible patterns of N=8 bits with up to M=4 rows and attempt to extrapolate the results to larger N-bit adder structures. The results will be studied and key findings may be extrapolated for larger N-bit adders.

The search algorithm was written in LabVIEW graphical programming language and all the code is attached in the Appendix. There are more details and descriptions of the LabVIEW code and subroutines included in the Appendices.

4 Results

A single desktop PC required almost 2 weeks to run the search algorithm without interruption and completed with the following amounts of prefix adder structures for N=8 bits with up to M=4 rows tallied in Table 2. A complete pattern represents a fully functional prefix adder structure and an incomplete pattern represents a prefix adder structure that does not have all the required group generate and propagate signals on all outputs. The incomplete pattern may potentially be made complete with additional row(s) of prefix logic. The tallies do not include structures that have idempotency. It is noteworthy to observe how the number of both complete and incomplete prefix structures increases exponentially at higher levels. There is no reason to run the search algorithm for depth beyond the $2 \log_2 N - 1$ upper limit established by Brent-Kung and Han-Carlson.

Table 2 – Tally of Parallel Prefix Adder Structures

Level (Row)	Complete Patterns	Incomplete Patterns
1	0	64
2	0	4160
3	52	172044
4	45786	4668266

4.1 Nodes, Fanout, and Depth Results

The delay, complexity, and fanout characteristics will be used as a first approach to compare all valid parallel prefix adder structures that have been generated by the search algorithm. Previous works have different weights on importance of what areas of optimization to focus on and how to best represent delay, complexity, and fanout. For the purpose of this first study, a simplistic approach will be taken by making the following approximations for comparison.

1. Complexity is represented by the number of nodes.
2. Fanout is represented by the product of the max fanout of any one branch and the number of nodes divided by the number of branches to indicate an average fanout measure. By capturing the worst-case fanout and an average in a single fanout metric, two prefix structures with the same average fanout can be readily distinguished from each other if

one has several smaller fanouts (more favorable) and another has a single large fanout (less favorable).

3. The delay is represented by the depth, i.e., number of rows or levels.

A straight-forward comparison approach is to perform a 2D plot of complexity versus fanout for each level. A 3D plot of delay, complexity, and fanout actually would be more interesting especially for larger N-bit parallel prefix adders, however, all the valid 8-bit parallel prefix structures that have been generated by the search algorithm for this work only existed with 2 delay points (3 or 4 levels) so it is simpler to examine the 2D plot at each level.

Graphing a single metric or namely the quality factor is another useful comparison approach. The quality factor is defined as follows:

$$\mathrm{QF}_0 = \text{Quality Factor} = (\text{Complexity})(\text{Fanout})(\text{Delay}) = \frac{(\text{Nodes})^2(\text{Max Fanout})(\text{Rows})}{\text{Branches}}$$

For example, the following table illustrates the characteristics for 8-bit parallel prefix adder structures.

Table 3 – Summary of Parallel Prefix Adder QF_0 Characteristics

Structure	Rows	Branches	Nodes	Max Fanout	QF_0
Kogge-Stone	3	17	17	1	51
Ladner-Fischer	3	7	12	4	252
Brent-Kung	4	10	11	2	96
Han-Carlson	4	12	12	1	48
Knowles (1,1,4)	3	11	14	4	212
Knowles (1,2,2)	3	9	14	2	130
Knowles(1,1,2)	3	15	17	2	116

Although the QF_0 definition is an overly simplified attempt at comparing parallel prefix structures and does not provide equal weight between Kogge-Stone, Ladner-Fischer, and Brent-Kung designs, the QF_0 metric is very useful to quickly screen out the thousands of valid prefix structures that are worse than the worst $QF_0 = 252$ of Ladner-Fischer.

4.1.1 3 Level 8-bit Prefix Adder Results

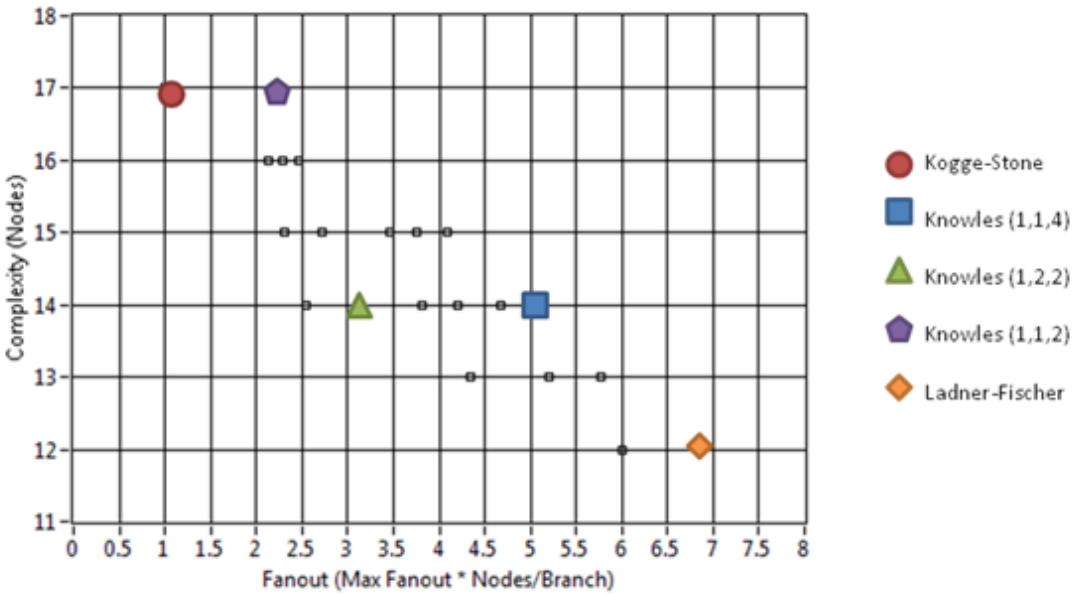
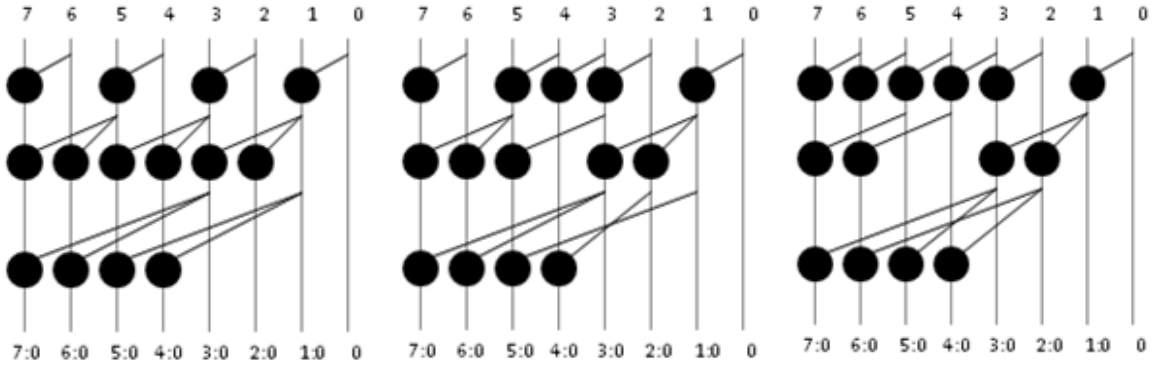


Figure 10 – 3 Level 8-bit Prefix Adder Complexity versus Fanout Plot

Two observations can be made from Figure 10:

1. There are only 21 points for 52 parallel prefix structures. This indicates that there are slightly different structures with redundant characteristics.
2. There is a nice array of tradeoff options between the two expected design extremes of Kogge-Stone and Ladner-Fischer.

Based on the graph in Figure 10, there are some prefix structures that appear slightly more optimal and closer to the graph origin than Knowles (1,2,2), particularly at the point (fanout, nodes) = (2.5, 14). There are 2 structures with this characteristic as follows in Figure 11.

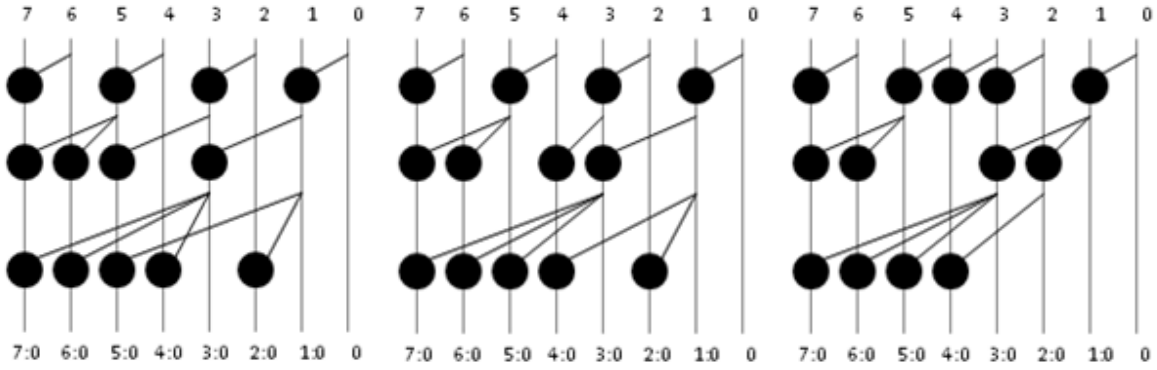


Knowles (1,2,2) *(2.5, 14) Structure 1* *(2.5, 14) Structure 2*

Figure 11 – Knowles (1,2,2) and (2.5, 14) Structures

Knowles (1,2,2) has 5 branches with fanout of 2, and the new structures above only have 3 branches with a fanout of 2.

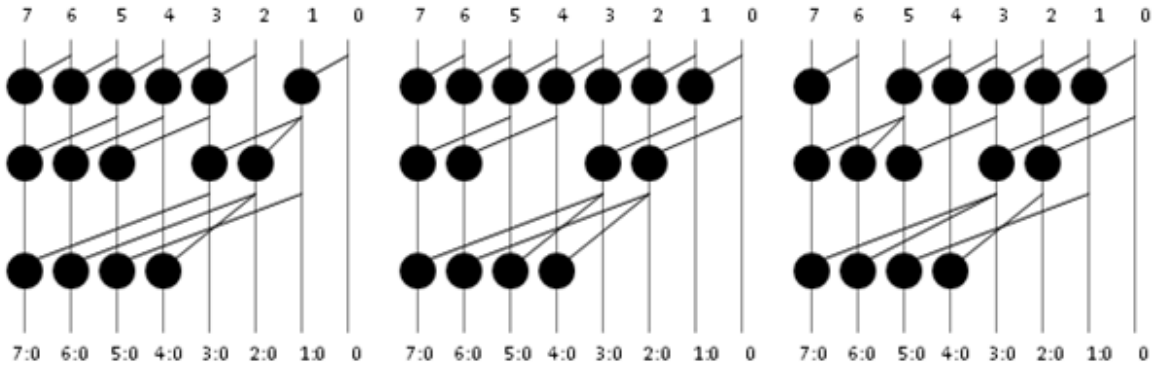
There are other structures with slightly more fanout or complexity such as at the points (4.3, 13) or (2.3, 15) in Figure 10. The search algorithm produced 5 structures with a (4.3, 13) characteristic, but the 3 most interesting ones are shown as follow in Figure 12:



(4.3, 13) Structure 1 *(4.3, 13) Structure 2* *(4.3, 13) Structure 3*

Figure 12 – (4.3, 13) Structures

The algorithm also produced 4 structures with (2.3, 15) characteristics, but only 3 interesting ones are shown as follows in Figure 13:



(2.3, 15) Structure 1 *(2.3, 15) Structure 2* *(2.3, 15) Structure 3*

Figure 13 – (2.3, 15) Structures

Overall one can observe and expect that all the 3 level 8-bit parallel prefix structures illustrated above exhibit various combined characteristics of Ladner-Fisher and Kogge-Stone. What is particularly interesting is the alternating prefix groupings especially at the 3rd row.

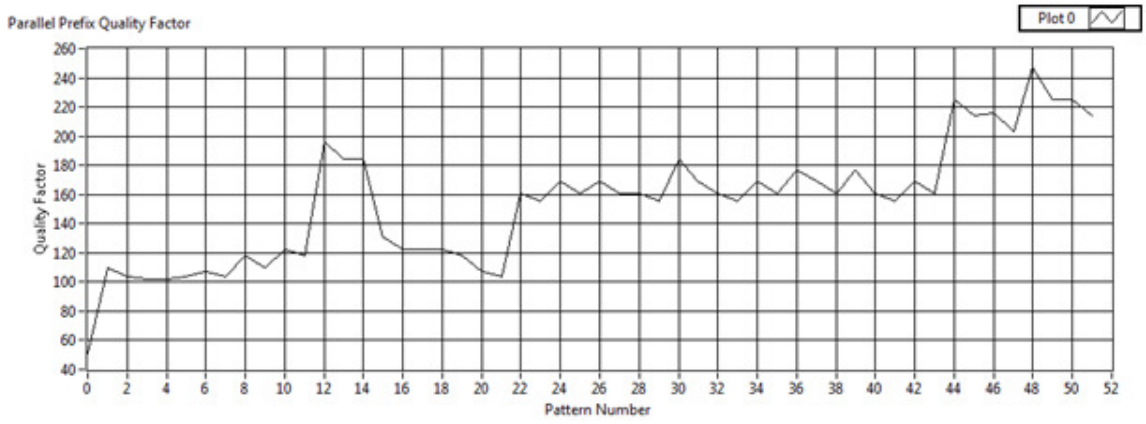


Figure 14 – 3 Level 8-bit Prefix Adder QF_0 Plot

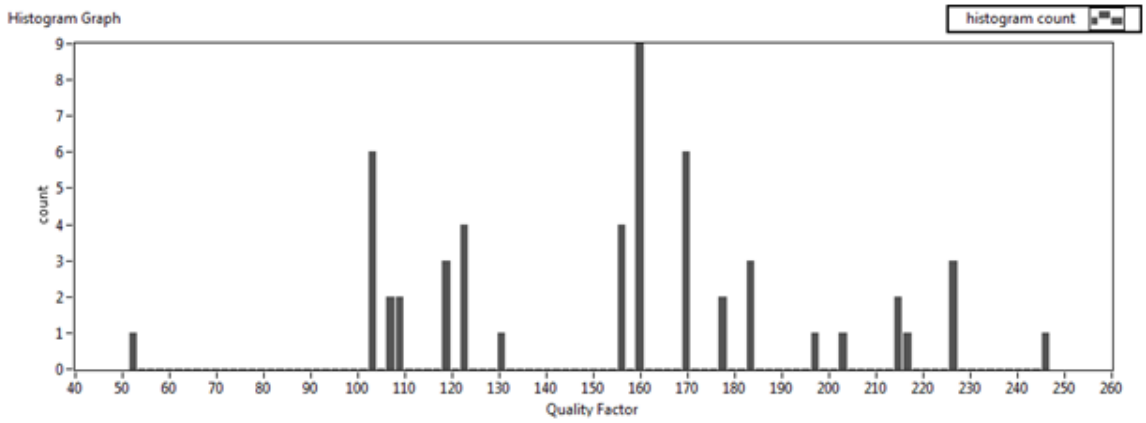


Figure 15 – 3 Level 8-bit Prefix Adder QF_0 Histogram

Of the 52 valid 8-bit prefix adder structures, the structure with the lowest QF_0 is Kogge-Stone (QF_0=51, pattern=0). The structures with the next lowest QF_0 = 100 are patterns 2, 3, 4, 5, 7, and 21. Pattern 2, 5, 7, and 21 are all (2.3, 15) structures, three of which are illustrated on the previous pages. Pattern 3 and 4 are (2.1, 16) structures, which is one node less than Kogge-Stone and has 1

branch with a fanout of 2. Overall, the QF_0 plot suggests that all 52 valid 8-bit prefix adder structures have qualities that fit in between Kogge-Stone (QF_0=51, pattern=0) and Ladner-Fischer (QF_0=252, pattern=48) and this finding supplements the observations made on Figure 10.

4.1.2 4 Level 8-bit Prefix Adder Results

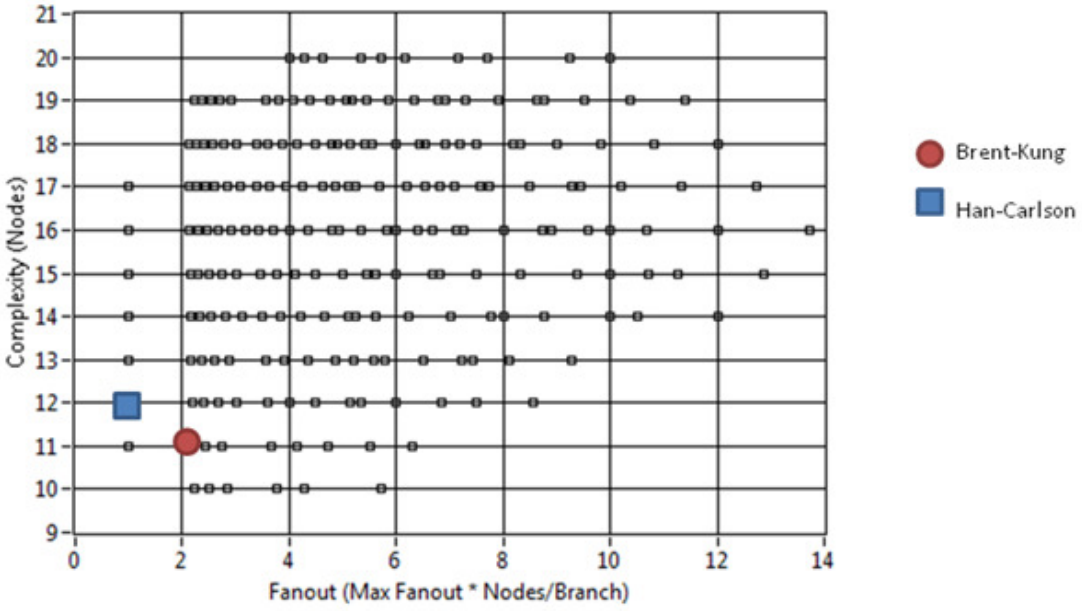
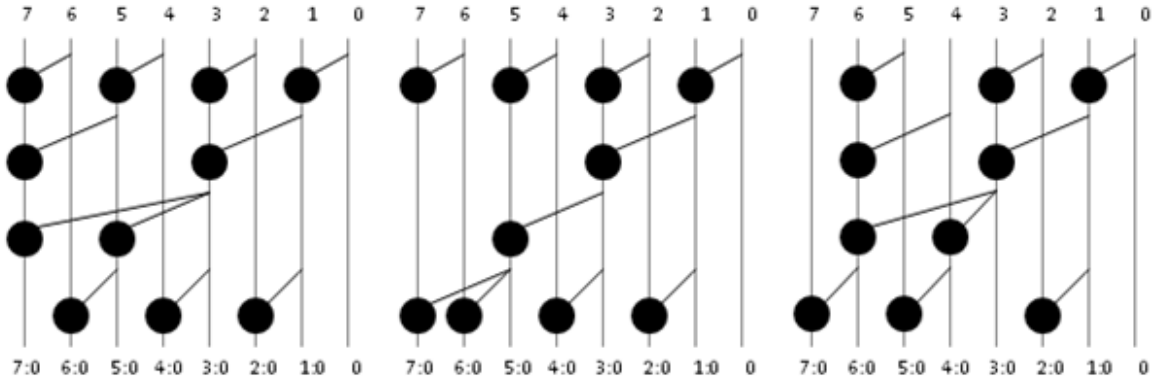


Figure 16 – 4 Level 8-bit Prefix Adder Complexity versus Fanout Plot

At 4 levels, the prefix adder is sacrificing minimum depth for a reduction in complexity. Brent-Kung and Han-Carlson have 11 and 12 nodes respectively for a 8-bit adder. The plot in Figure 16 shows that Brent-Kung and Han-Carlson are among the most optimal minimum complexity structures possible located closest to the origin, but the search algorithm uncovered several other structures that are slightly more optimal than Brent-Kung at the point (fanout, nodes) = (1, 11) and also at (2.2, 10).

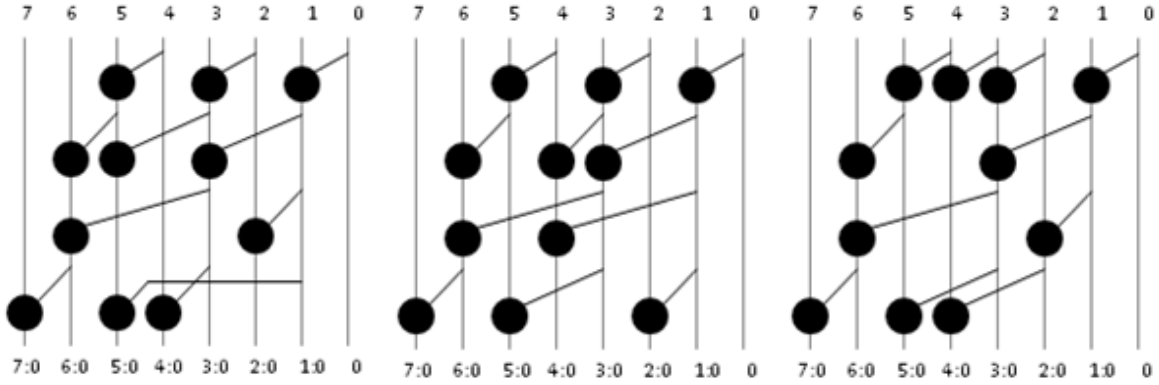
An interesting subset of those new structures are illustrated as follows in Figure 17.



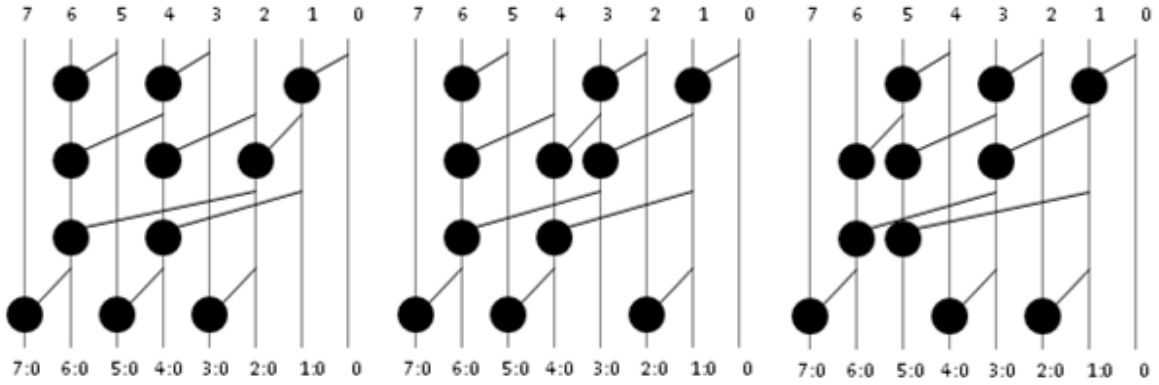
Brent-Kung *(2.2, 10) Structure 1* *(2.2, 10) Structure 2*

Figure 17 – Brent-Kung and (2.2, 10) Structures

The key observation of the (2.2, 10) structures is that it saves one node on the MSB column with a few nodes rearranged.



(1, 11) Structure 1 *(1, 11) Structure 2* *(1, 11) Structure 3*



(1, 11) Structure 4 *(1, 11) Structure 5* *(1, 11) Structure 6*

Figure 18 – (1, 11) Structures

The (1, 11) structures do not have a clear systematic design pattern, but the key takeaway is that a non-standard structure can be slightly more optimal. The advantage the (1,11) structures have over Brent-Kung is that all branches have no fanout to multiple nodes and the advantage over Han-Carlson is 1 less node (slightly lower hardware cost).

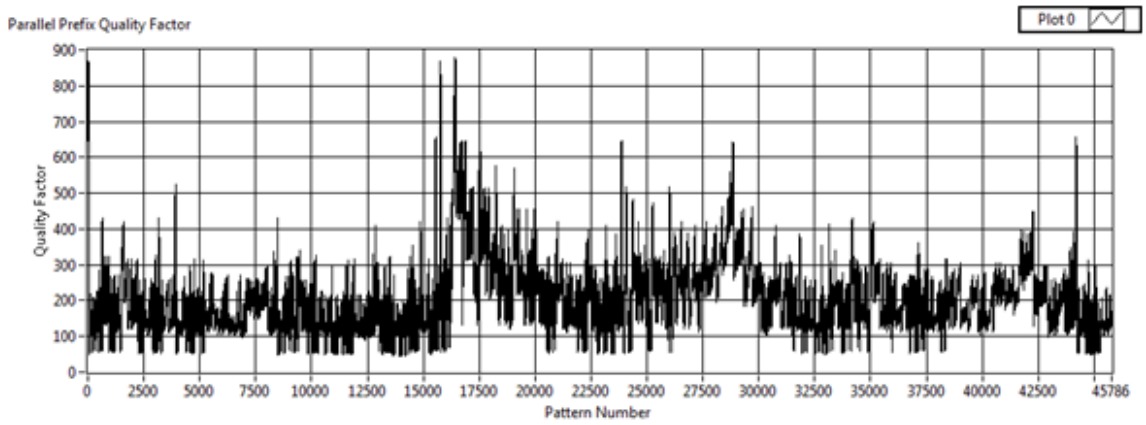


Figure 19 – 4 Level 8-bit Prefix Adder QF_0 Plot

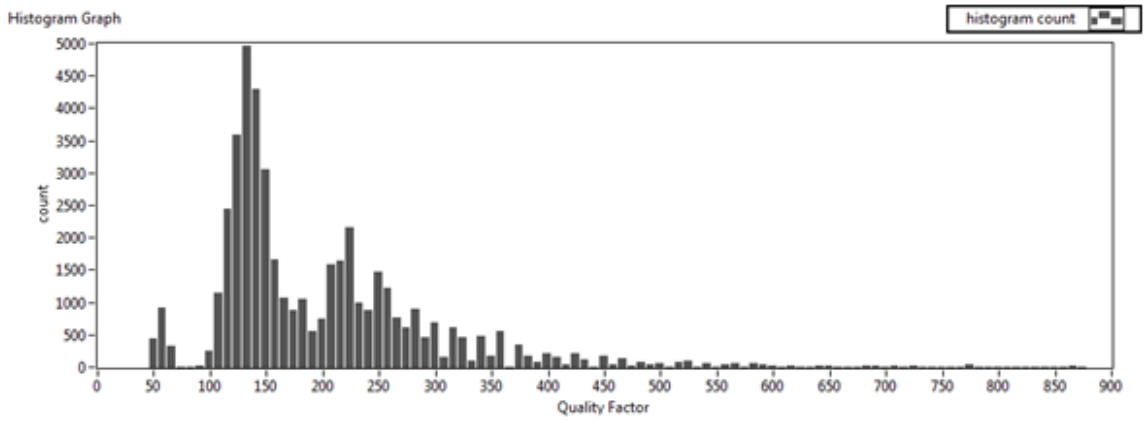


Figure 20 – 4 Level 8-bit Prefix Adder QF_0 Histogram

Of the 45786 valid 8-bit prefix adder structures, Figure 20 shows that there are at least 1500 possible 8-bit patterns with a QF_0 between Han-Carlson ($QF_0 = 48$) and Brent-Kung ($QF_0 = 96$).

4.1.3 Extrapolation to Larger N-bit Min Depth Parallel Prefix Adder Structures

The 3-level 8-bit (2.5, 14) Structure 1 and (2.3, 15) Structure 2 patterns are very interesting to extrapolate to larger N-bit adders because they exhibit a regular structure. Figures 21 and 22 illustrate 16-bit prefix structures based on the 8-bit structures.

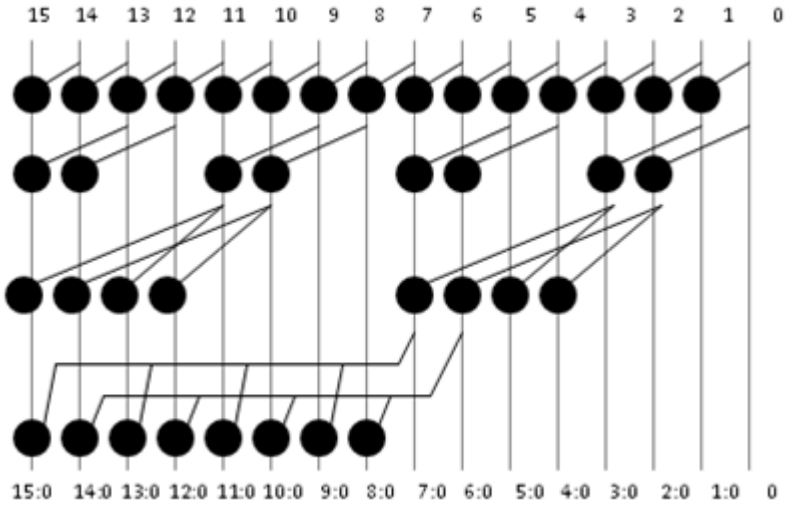


Figure 21 – (Structure A) – 4 Level 16-bit based on 3-level 8-bit (2.5, 15) structure

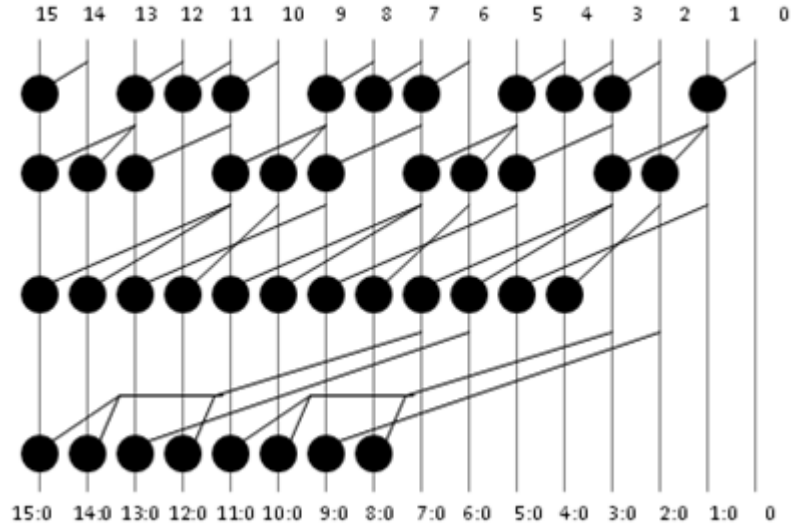


Figure 22 – (Structure B) – 4 Level 16-bit based on 3-level 8-bit (2.4, 14) structure

4.1.4 Comparison of Minimum Depth (4 level) 16-bit Prefix Adders

Table 4 – Comparison of Minimum Depth 16-bit Prefix Adder Characteristics

Structure	Nodes	Branches	Max Fanout	Fanout Metric
Kogge-Stone	49	49	1	1
Knowles (1,1,1,4)	49	43	4	4.6
Structure B	42	31	3	4.1
Knowles (1,1,2,4)	42	30	4	5.6
Knowles (1,1,4,4)	40	25	4	6.4
Knowles (1,2,2,4)	42	23	4	7.3
Structure A	39	29	4	5.4
Knowles (1,2,4,4)	36	17	4	8.5
Ladner-Fischer	32	15	4	8.5

Structure A is noteworthy because it fits in between Knowles (1,2,2,4) and Knowles (1,2,4,4) for complexity, but has a higher branch count because it has more individual branches and thus exhibits a lower fanout metric that is more similar to Knowles (1,1,2,4). Essentially, Structure A increases the number of individual branches while keeping the complexity lower than a comparable Knowles prefix structure.

Structure B is nearly identical to a Knowles (1,1,2,4) in terms of complexity, but has a max fanout of 3 rather than 4, so it is able to achieve a fanout metric that is lower than Knowles (1,1,1,4) with less complexity.

There are several other larger N-bit parallel prefix structures that could potentially be extrapolated from the 8-bit results.

4.1.5 Optimizing Larger N-bit Brent-Kung Prefix Adder Structures

The 4-level 8-bit (2.2, 10) Structure 1 can be easily extended to larger Brent-Kung adders by saving one node as illustrated in Figure 23.

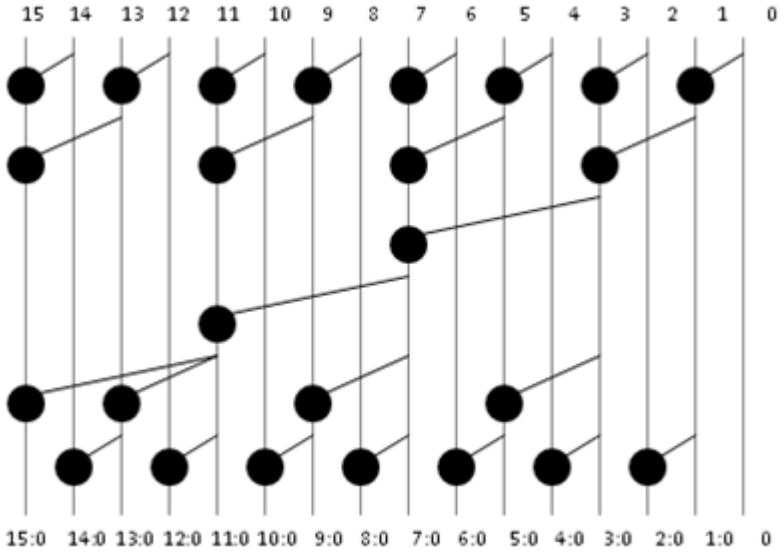


Figure 23 – (Structure C) – 5 Level 16-bit based on 4-level 8-bit (2.2, 10) structure

The minor modification has the benefit of eliminating one node on the MSB and reducing the number of wire crossings for the final node on the MSB. The benefits diminish for larger N-bit adders.

The 4-level 8-bit (1, 11) structures are slightly more optimal than Brent-Kung, but due to the non-standard structure they are hard to extrapolate to larger N-bit parallel prefix structures. Nonetheless, this work shows that slightly more optimal non-standard structures do exist.

4.2 Area, Power, and Delay Results

The first study involved a relatively simplistic approach for comparing characteristics of different adders. Complexity was represented by the number of nodes. Fanout was represented by the product of the max and average fanout. Delay was represented by the number of levels. The next few sections describe the shortcomings of the first study and how it can be refined.

Fanout is not usually a direct design parameter in digital circuit design. Normally gate sizing and fanout constraints are driven by area, speed, and power requirements. Therefore, a study of the fanout characteristics of different parallel prefix adders is not necessarily directly useful.

Using the depth of a parallel prefix adder as a delay metric is very simplistic and assumes all nodes have the same average unit "node" delay and neglects the effect of fanout and wire capacitances. Using a logical effort model approach will improve the delay distinction between adder structures because it takes in account the effect of fanout on nodes in the critical signal path. The critical signal path starts from the LSB input of the adder to the deepest output bit, which is usually the MSB or the bit below it. There may be more than one path of the same depth so the critical path will be the worst-case logical effort of these paths.

The logical effort model described in [7] and [8] enables computation of the minimum possible path delay D as follows:

$$D = NF^{(1/N)} + P$$

1. $F = GBH$ = overall path effort
2. $G = \prod g_i$ = path logical effort where g_i is the logical effort of each intermediate part
    a. Logical effort = $C_{in\_device} / C_{unit\_inverter}$
3. $B = \prod b_i$ = path branch effort where b_i is the branch effort in each intermediate stage
    a. Branch effort = $(C_{on\_path} + C_{off\_path}) / C_{on\_path}$
4. $H = C_{out}/C_{in}$ = path electrical effort or ratio of output to input capacitance for the circuit
5. $P = \sum p_i$ = path parasitic delay where p_i is the parasitic delay of each stage

Therefore, the minimum possible delay D can be computed based on the following:

1. G will be neglected and equal to 1 because the logical effort for a buffer and a node is about the same.
2. B is computed as the product all the fanouts on the critical path when assuming the buffer and all node inputs have similar input capacitance. This simplification helps in analyzing the large data set of results, although including some measure of wire capacitance and distinction between different input capacitances of nodes and buffer circuits would help improve the accuracy of the results.
3. H will be neglected and equal to 1 because the input and output capacitances of the prefix adder structure are the same.
4. P is computed as the sum of the parasitic delays of individual nodes. This can vary depending on whether the path goes through a buffer or a certain input in the group logic node, so for the purpose of this work to simplify the comparison, all paths are assumed to have approximately the same parasitic delay so the value of P will be neglected and assumed to be 0.

In summary, the refinement to the delay metric is simply based on the product of the branches on the critical path. As a result, the delay expression nicely takes in account the parallel prefix adder depth, effects of fanout and characteristics of the actual critical path all in a single metric and essentially reduces 2 characteristic dimensions (depth and fanout) of the original study approach into a single dimension (delay). A small algorithm was developed to analyze each parallel prefix structure to identify critical paths and compute the worst-case delay based on the aforementioned assumptions. The algorithm is described in Appendix B.

Furthermore, it is important to point out that all discussions up until this point assumed the fanout count did not include the implicit vertical (column) branch between buffers or nodes, that is, fanout only counted actual branches to new nodes. From this point onward, the fanout count will now include implicit branches mainly because the critical path especially in non-minimum depth parallel prefix adder structures do not always traverse through nodes. For instance, the 16-bit Han-Carlson in Figure 5 has two possible critical paths that are highlighted in Figure 24 with 5 branches at columns starting and ending as 0→1→3→7→15→15 or 0→1→1→5→13→14, and these paths have branch efforts of 16 and 32 respectively. The latter path has an implicit branch in bit column 1.

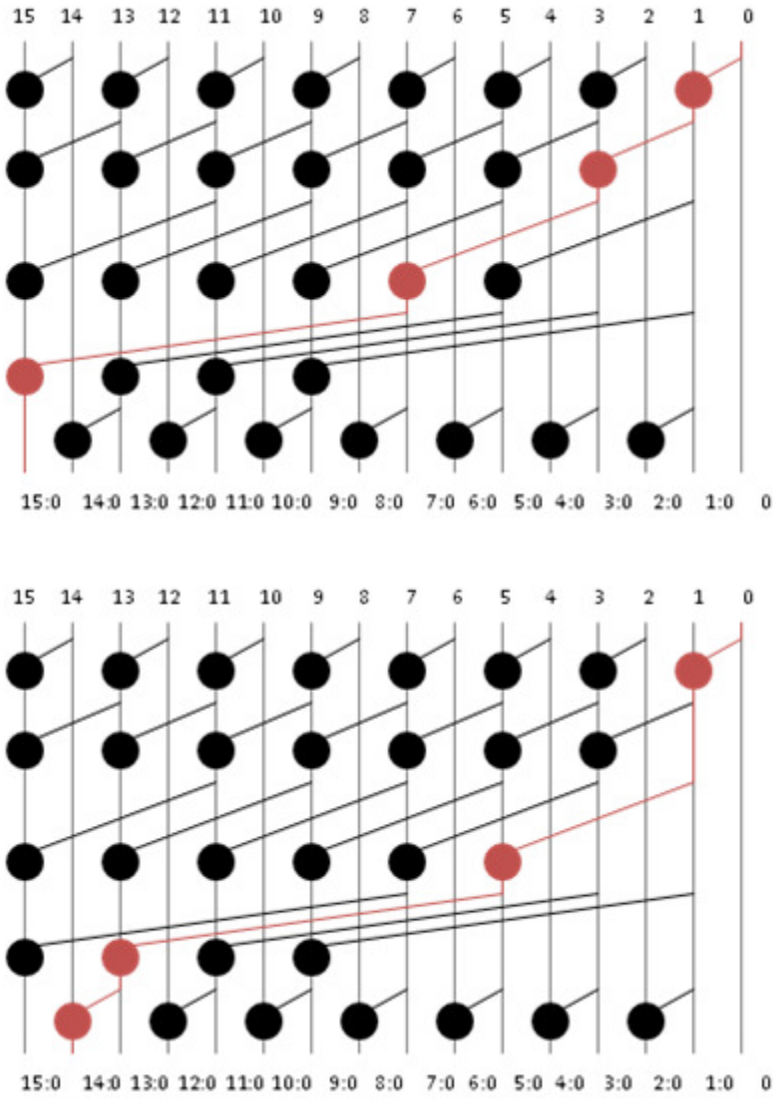


Figure 24 – 16-bit Han-Carlson prefix adder propagation delay paths

A complexity metric that only relies on the number of nodes in an adder structure leaves out the impact of creating interconnections between nodes. For example, two structures may have an identical number of nodes, but one has a greater amount of wires compared to the other and thus contributes to larger area. A more faithful estimate of area is challenging to predict and compare among parallel prefix adders because of the variation in node placement and irregularity in their interconnections. However, representing area as the product of total number of nodes and total wire length would be generally an improvement in distinguishing adder designs. Total wire length is calculated as the sum of all horizontal distances of all wires from source to destination. The vertical distances of all wires are assumed to be negligible relative to horizontal wire lengths.

Power dissipation was not a focus in the first study approach. Low power consumption has become increasingly important in digital design because it is motivated by demand for mobile devices and silicon cooling challenges due to increasing power density caused by shrinking transistor sizes. Additionally, as CMOS technology continues to shrink, static power dissipation due to leakage has been trending towards a larger share of total power dissipation, so the number of nodes would be considered a good metric of static power dissipation because more nodes, i.e., transistors, dissipate more static power regardless of whether a signal is propagating through the node. The second aspect which was neglected in the first study approach is dynamic power. Dynamic power could be generally modeled based on the capacitances and switching probabilities of each type of logic gate in the design, however, it becomes more challenging for a binary adder because different architectures exhibit different switching propagation paths that depend on the input values [9]. It is useful, however, to recognize that the dynamic power contribution could be generally approximated by the total wire length, i.e., total wire capacitance, in the adder structure.

As a result, all parallel prefix adder structures could be compared using an area-power metric and delay. The Quality Factor could also be defined as follows:

$$\mathrm{QF}_1 = \text{Quality Factor} = (\text{Area})(\text{Power})(\text{Delay}) = (\text{Nodes})(\text{Total Wire Length})(\text{Delay})$$

Table 5 summarizes the area-power and delay metrics as well as the QF_1 using the aforementioned approach for existing adder structures.

Table 5 – Summary of 8-bit Prefix Adder Characteristics

Structure	Nodes	Wire Length	Area·Power	Branch Effort	Levels	Delay	QF_1
Kogge-Stone	17	51	595	8	3	6.00	3570
Knowles(1,1,2)	17	25	425	12	3	6.87	2920
Knowles (1,2,2)	14	27	378	18	3	7.86	2971
Knowles (1,1,4)	14	24	336	20	3	8.14	2735
Han-Carlson	12	21	252	16	4	8.00	2016
Ladner-Fischer	12	20	240	30	3	9.32	2237
Brent-Kung	11	17	187	24	4	8.85	1655

The Area·Power metric has a stronger effect on the QF_1 metric than the Delay metric. Although it is readily possible to apply a linear adjustment to balance the contributions, the purpose of the QF_1 metric, like the QF_0 metric, is only to quickly compare and screen out the thousands of valid prefix structures that are worse than the worst $QF_1 = 3570$ of Kogge-Stone.

4.2.1 8-bit Prefix Adder Results

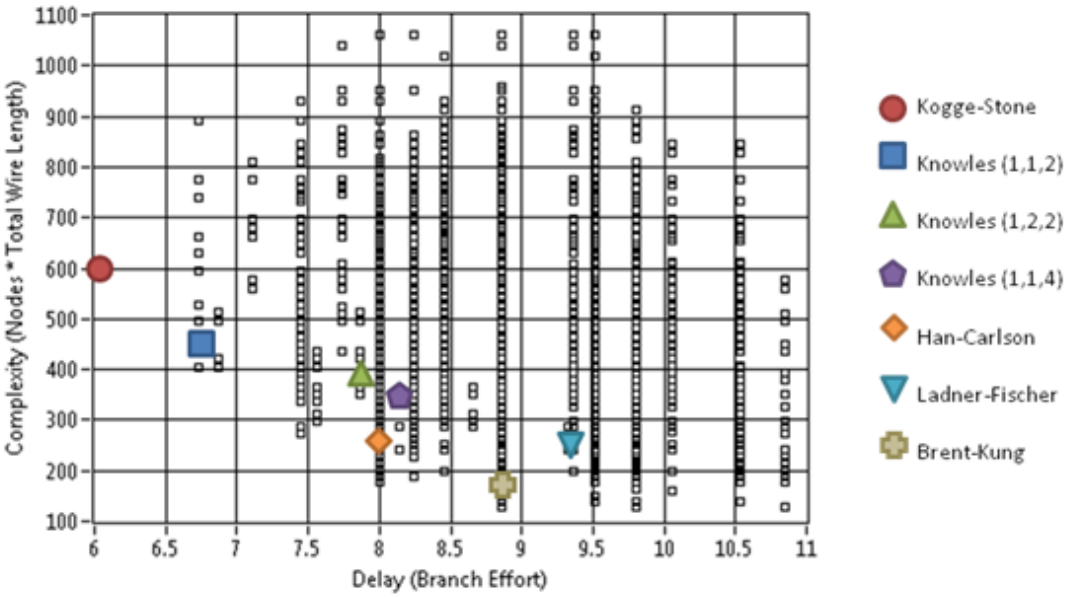


Figure 25 – 8-bit Parallel Prefix Adder Area·Power versus Delay Plot

Figure 25 nicely illustrates a complete spectrum of 8-bit parallel prefix adder characteristics in a single graph including both minimum depth and minimum complexity structures. The adder depth and fanout are now combined into a normalized delay metric. It is notable to observe that the graph still follows a similar pattern as in the first study approach (Figures 10 and 16) that examined characteristics in 3 areas (nodes, fanout, and depth). At one end of the spectrum, Kogge-Stone is the fastest and higher cost, and at the other end is Ladner-Fischer, which is slower and lower cost. Han-Carlson and Brent-Kung exhibit lower delay than Ladner-Fischer and some of the Knowles adders because of their lower branch effort on the critical path. These observations are also roughly in line with the remarks and conclusions in [8]. Lastly, an important observation of Figure 25 is that there indeed exist parallel prefix adder structures that are more efficient than the well-documented designs and supplements the findings of the first study approach in Section 4.1.

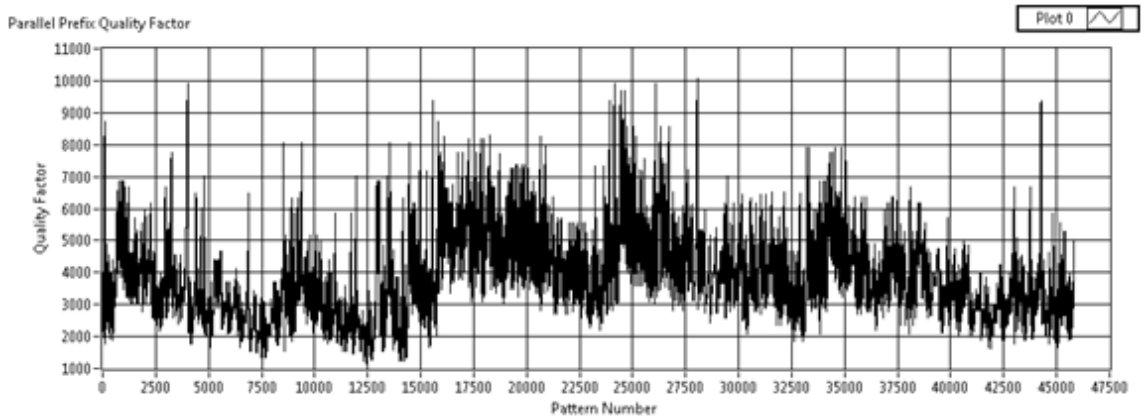


Figure 26 – 8-bit Parallel Prefix Adder QF_1 Plot

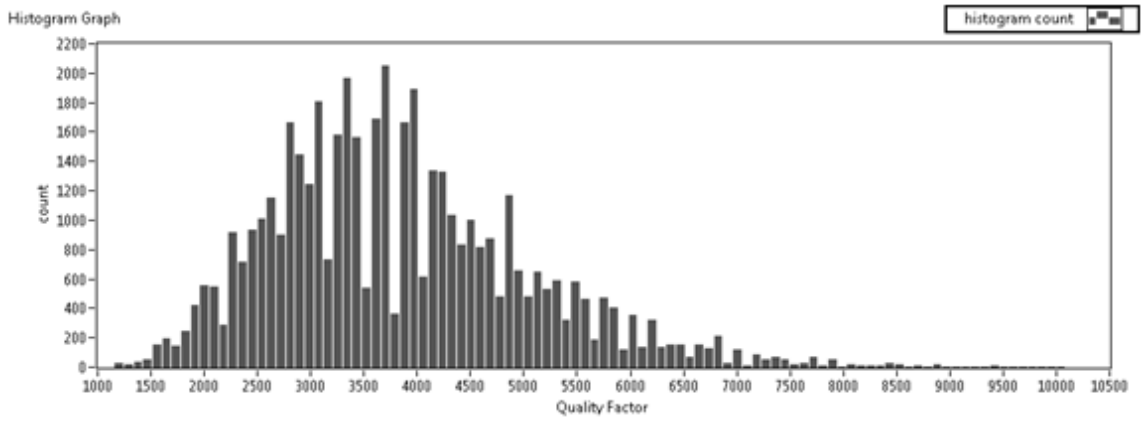


Figure 27 – 8-bit Parallel Prefix Adder QF_1 Histogram

To further supplement the observations on Figure 25, Figure 27 shows that there exist at least 100 prefix adder structures with a QF_1 metric better than the lowest QF_1 = 1655 of Brent-Kung. A majority of the prefix adder structures have a QF_1 that lie between 2500 to 5000, and Kogge-Stone QF_1 = 3570 lies roughly in the middle of the histogram curve. Also, Figure 25 showed all the 8-bit parallel prefix adders and suggested that the minimum depth adder structures are not necessarily the most efficient overall especially when the structure trends towards higher branch efforts. For comparison to Figure 10, only the 3 level prefix adder Area·Power versus Delay plot are shown in Figure 28.

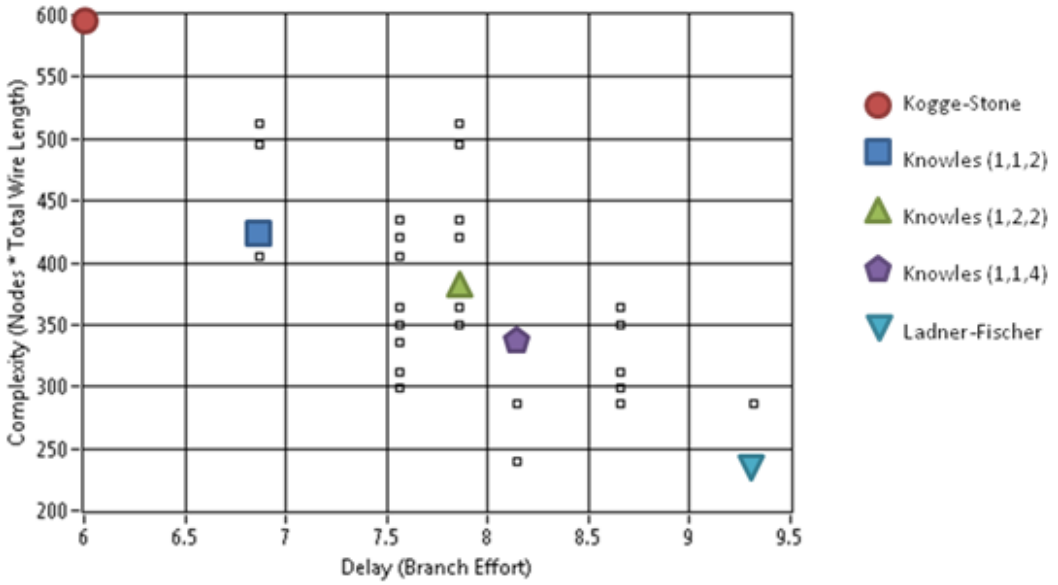
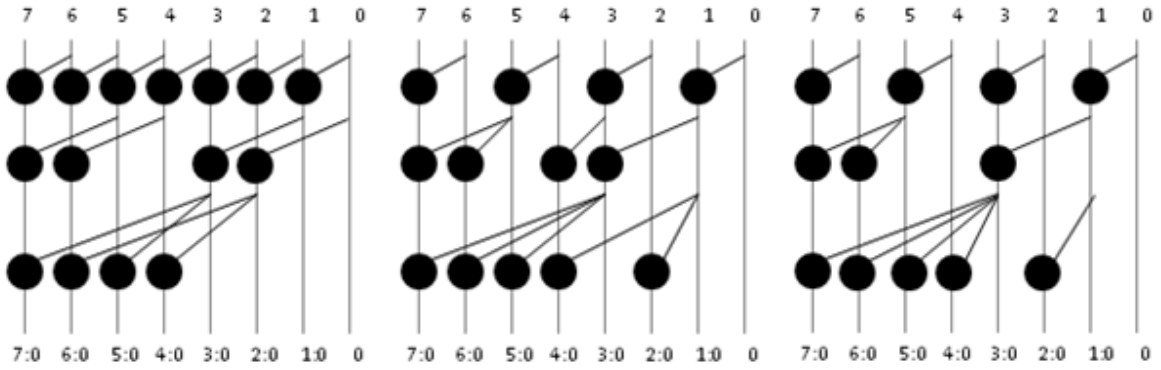


Figure 28 – 3 Level 8-bit Parallel Prefix Adder Area-Power versus Delay Plot

Figure 28 shows similar characteristics as Figure 10. Kogge-Stone and Ladner-Fischer lie on the extreme ends of the spectrum and the Knowles class of adder structures lie in between. There are adder structures that are more efficient and lie at the points (delay, complexity) = (6.87, 405), (7.56, 299), and (8.14, 240) and these structures are illustrated in Figure 29.



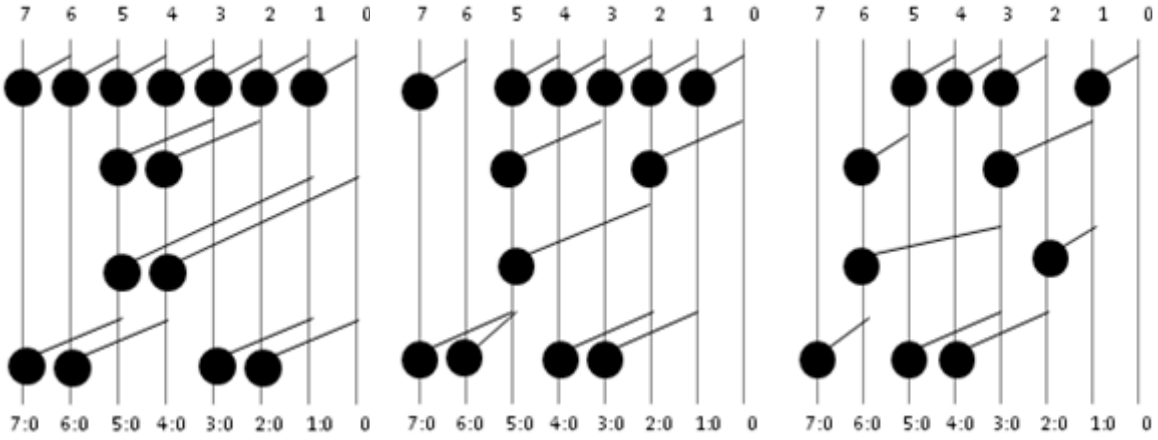
(6.87, 405) Structure *(7.56, 299) Structure* *(8.14, 240) Structure*

Figure 29 – Area-Power-Delay Efficient 3 Level 8-bit Structures

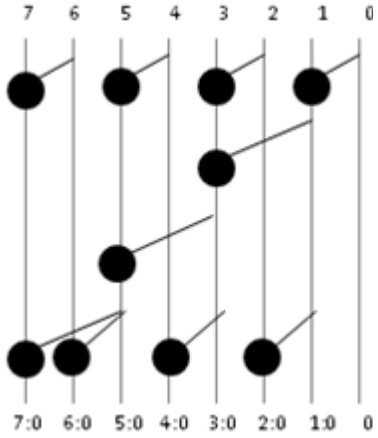
Observations based on Figure 29:

1. The (6.87, 405) structure is identical to the (2.3, 15) Structure 2 shown in Figure 13 and has been extrapolated to a 16-bit Structure B in Section 4.1.3. An improved variation that is also based on the observations of (8.14, 240) structure is shown in Section 4.2.2.
2. The (7.56, 299) structure is identical to the (4.3, 13) Structure 2 shown in Figure 12.
3. The (8.14, 240) structure is a subtle modification of the Ladner-Fischer structure by moving the last node in Column 2 down from the 2nd to 3rd row. The key improvement of this modification is the branch effort of the critical path is reduced from 30 to 20, and therefore the delay is reduced from 9.32 to 8.14, which is a significant improvement. Also, this improvement extends very well to larger N bit adders and a 16-bit version is illustrated in Section 4.2.2.

However, as the plot of all 8-bit parallel prefix adder characteristics in Figure 25 indicated, the minimum depth prefix structures are not necessarily the most efficient because of the larger branch efforts on the critical path and higher hardware costs (total wires and nodes). The efficient structures in Figure 24 that are closer to the origin reside along the points (6.73, 405), (7.44, 286), (8.00, 176), and (8.85, 130). The most interesting of these structures are illustrated in Figure 30.



(6.73, 405) Structure *(7.44, 286) Structure* *(8.00, 176) Structure*



(8.85, 130) Structure

Figure 30 – Area-Power-Delay Efficient 8-bit Prefix Structures

Observations based on Figure 30:

1. The (6.73, 405) structure exhibits slightly better delay than the minimum depth (6.87, 405) structure identified earlier because it has a lower branch effort on the critical path by moving nodes for the lower bits deeper in the structure. It also exhibits similar characteristics of staggering the prefix groupings as was observed in the (6.87, 405) structure.

2. The (7.44, 286) and (8.00, 176) structures are hybrids containing characteristics of other adders, but they exhibit slightly better area-power and delay characteristics compared to the nearest efficient minimum depth structure. However, they are difficult to extrapolate to larger N-bit adders.
3. The (8.85, 130) structure is identical to the (2.2, 10) Structure 1 identified in Section 4.1.2 and has been extrapolated to a 16-bit Structure C in Section 4.1.5. It has the same delay characteristic as Brent-Kung, but with a significantly lower area-power metric due to fewer nodes.

4.2.2 Extrapolation to Larger N-bit Parallel Prefix Adder Structures

The modified Ladner-Fischer that led to the 8-bit (8.14, 240) structure in Figure 29 is easy to extend to 16-bits. The key lesson here is to minimize the branch effort on the critical path by moving the lower bit branches further down the structure and inserting buffers in their place. This allows the adder attain a better branch effort on the critical path than Brent-Kung while maintaining minimum depth.

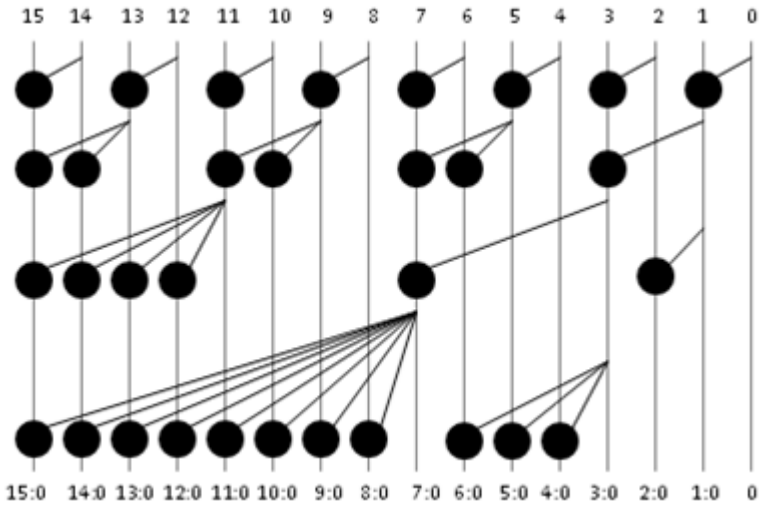


Figure 31 – Modified Ladner-Fischer based on (8.14, 240) Structure

Furthermore, the useful modification that was applied to Ladner-Fischer can also be applied to the (6.87, 403) structure that has already been extrapolated to 16-bits in Section 4.1.3. Figure 32 shows Structure A (Section 4.1.3) with an improved branch effort.

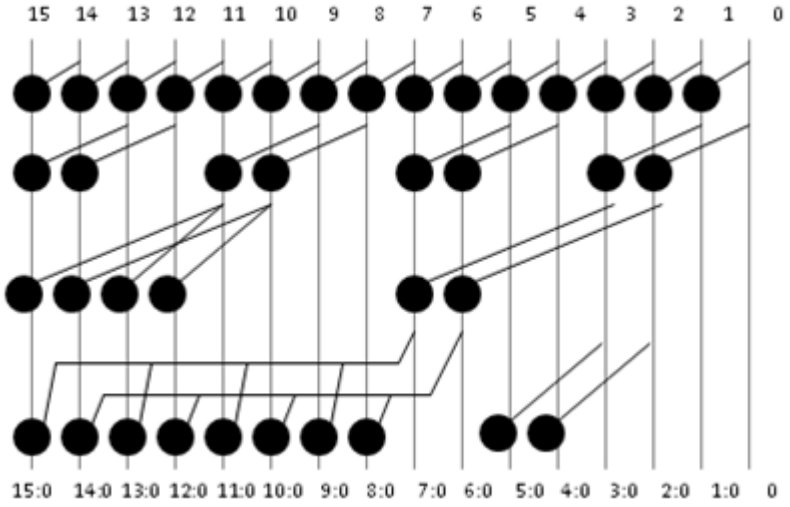


Figure 32 – Modified Structure A based on (6.87, 403) and (8.14, 240) Structures

Figure 33 illustrates a Structure D based on the design patterns observed in the (6.73, 405) and (6.87, 405) structures. Without the modification applied to Structure A as illustrated in Figure 28, Structure D has a slight edge over Structure A in that it has a lower branch effort. There is potentially a better solution than what was proposed for the 5th row to achieve the required group signals to further reduce the branch effort.

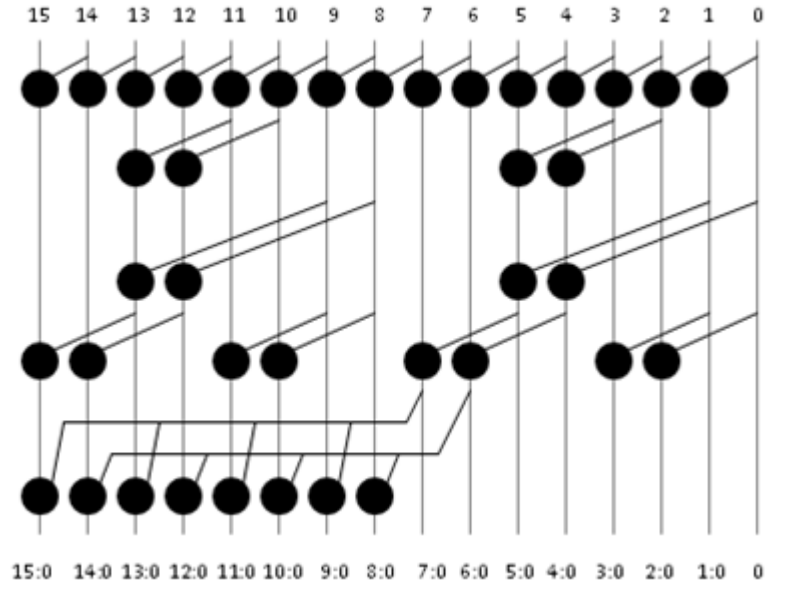


Figure 33 – (Structure D) – 16-bit structure based on (6.73, 405) and (6.87, 405) Structures

4.2.3 Comparison of 16-bit Prefix Adders

Table 6 – Summary of 16-bit Prefix Adder Characteristics

Structure	Nodes	Wire Length	Area·Power	Branch Effort	Levels	Delay
Kogge-Stone	49	155	7595	16	4	8.00
Knowles (1,1,1,4)	49	143	7007	40	4	10.06
Knowles (1,1,2,4)	42	116	4872	60	4	11.13
Knowles (1,2,2,4)	42	115	4830	90	4	12.32
Knowles (1,1,4,4)	40	110	4400	100	4	12.65
Structure D	39	95	3705	40	5	10.45
Structure A	39	89	3471	60	4	11.13
Modified Struct A	39	89	3471	40	4	10.06
Knowles (1,2,4,4)	36	102	3672	150	4	14.00
Han-Carlson	32	85	2720	32	5	10.00
Ladner-Fischer	32	76	2432	270	4	16.21
Modified L-F	32	76	2432	72	4	11.65
Brent-Kung	26	49	1274	96	5	12.46
Structure C	25	41	1025	96	5	12.46

Table 6 summarizes the area-power and delay metrics for the existing adder structures and the extrapolated structures developed from the first study approach (Figures 21 and 22).

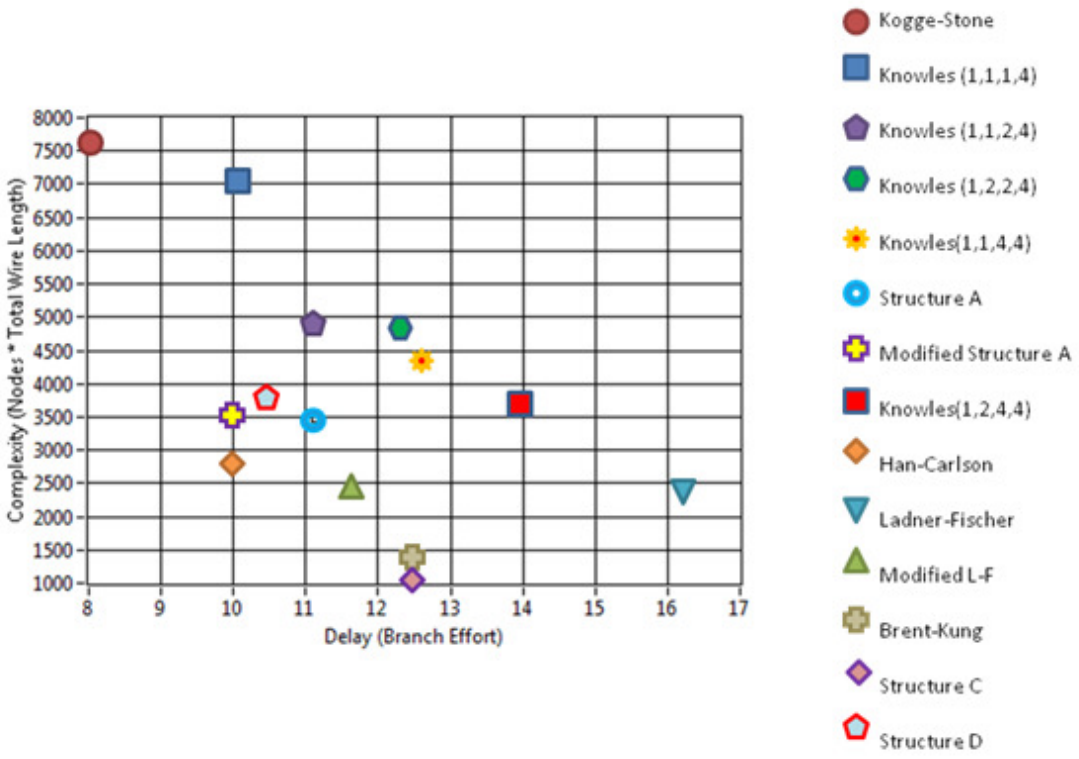


Figure 34 – 16-bit Parallel Prefix Adder Area-Power versus Delay Plot

Figure 34 is a graphical representation of Table 6 and illustrates the spectrum of 16-bit prefix adder structures. The existing prefix adders still remain roughly in the same place relative to each other for 16-bits relative to 8-bits, but more remarkably, Han-Carlson and Brent-Kung appear to grow less quickly in delay relative to Ladner-Fischer as the adder size grows. The modification on Ladner-Fischer and Structure A to reduce the branch effort on the critical path brings the delay on par with Han-Carlson and Brent-Kung, but the improvement will be challenging to maintain as the adder size grows.

5 Conclusion

Overall, the key takeaway from the new minimum-depth parallel prefix structures that have been uncovered by the search algorithm is that lower complexity and fanout metric results can be attained by staggering the prefix operator groupings as illustrated in Section 4.1.3. This increases the wire crossings by the branches that are not seen in any of the known documented prefix adder structures. Modern day silicon fabrication is capable of handling several metal layers so these new structures have potential for better speed performance and lower area compared to other minimum depth prefix adder structures.

For the minimal complexity parallel prefix structures, Han-Carlson and Brent-Kung are fairly close to optimal with the benefit of an easily scalable design pattern. These adders also exhibit very appealing speed qualities as the adder size grows in size because their branch efforts double as size doubles, unlike Knowles and Ladner-Fischer whose branch efforts grow exponentially as size doubles. Also this work demonstrated that there are non-standard prefix structures that are slightly more optimal for complexity and fanout, but the effort to find these structures for larger N-bit adders may not be worth the small return in performance depending on the application.

One area that was not explored with the search algorithm is allowing idempotency. This work did not explore this because the number of possible prefix structures increases significantly when allowing idempotency. However, this work compared the results to various Knowles adders that have redundancy and illustrated that there are inefficiencies with allowing idempotency.

The Area·Power and Delay study could be extended with further refinements to the metrics for improving the comparison of results with the ultimate goal of better quantifying the most optimal prefix adder structure for a 8-bit adder and potentially larger N-bit adders. The delay estimate could be refined as described in [7] and [8].

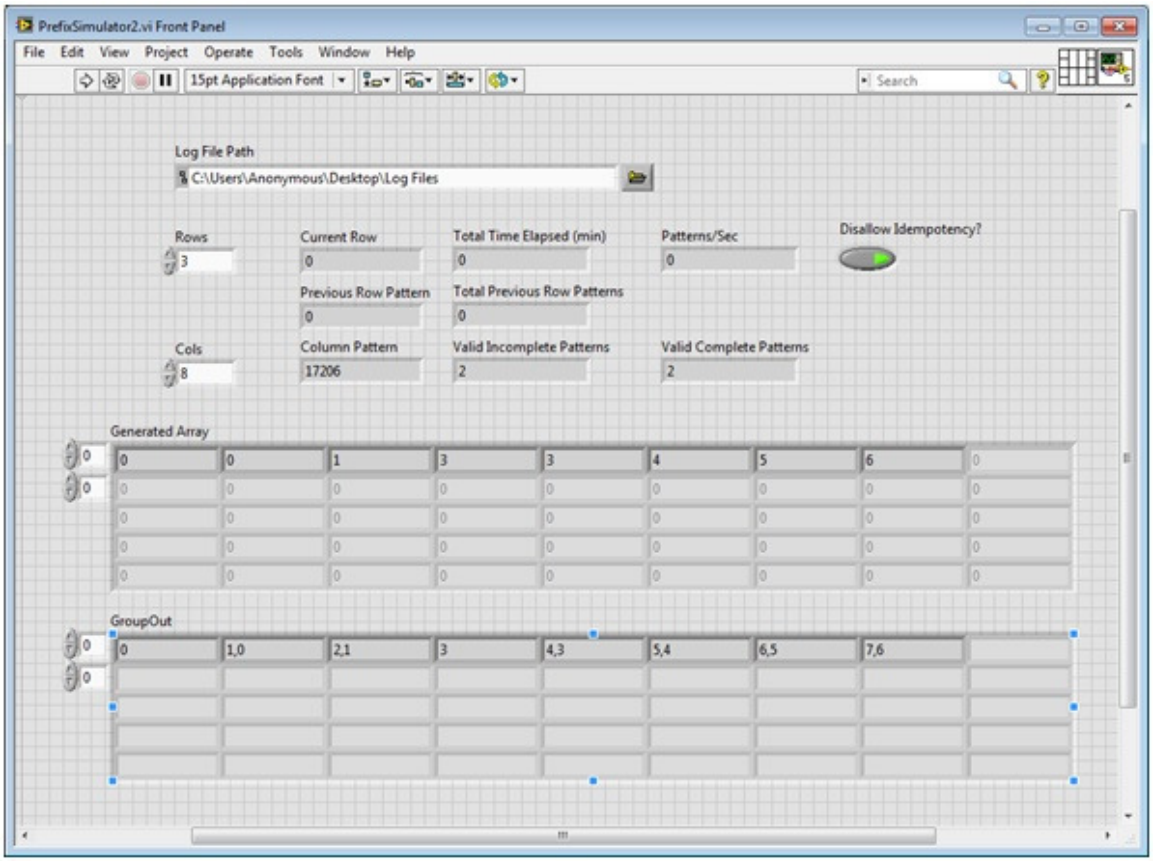
This search algorithm work could also be continued to search for optimal structures in 16, 32, or 64 bit prefix adders. The current algorithm design described in this work requires no modifications to work with larger adder sizes, however, the search time increases exponentially and was not feasible to run the search algorithm on a standard desktop computer and complete within reasonable time. It may be possible to execute the search algorithm faster on higher

performance computers and gather results within reasonable time. There are potentially other search algorithm optimization tricks that could be applied to speed up the search time.

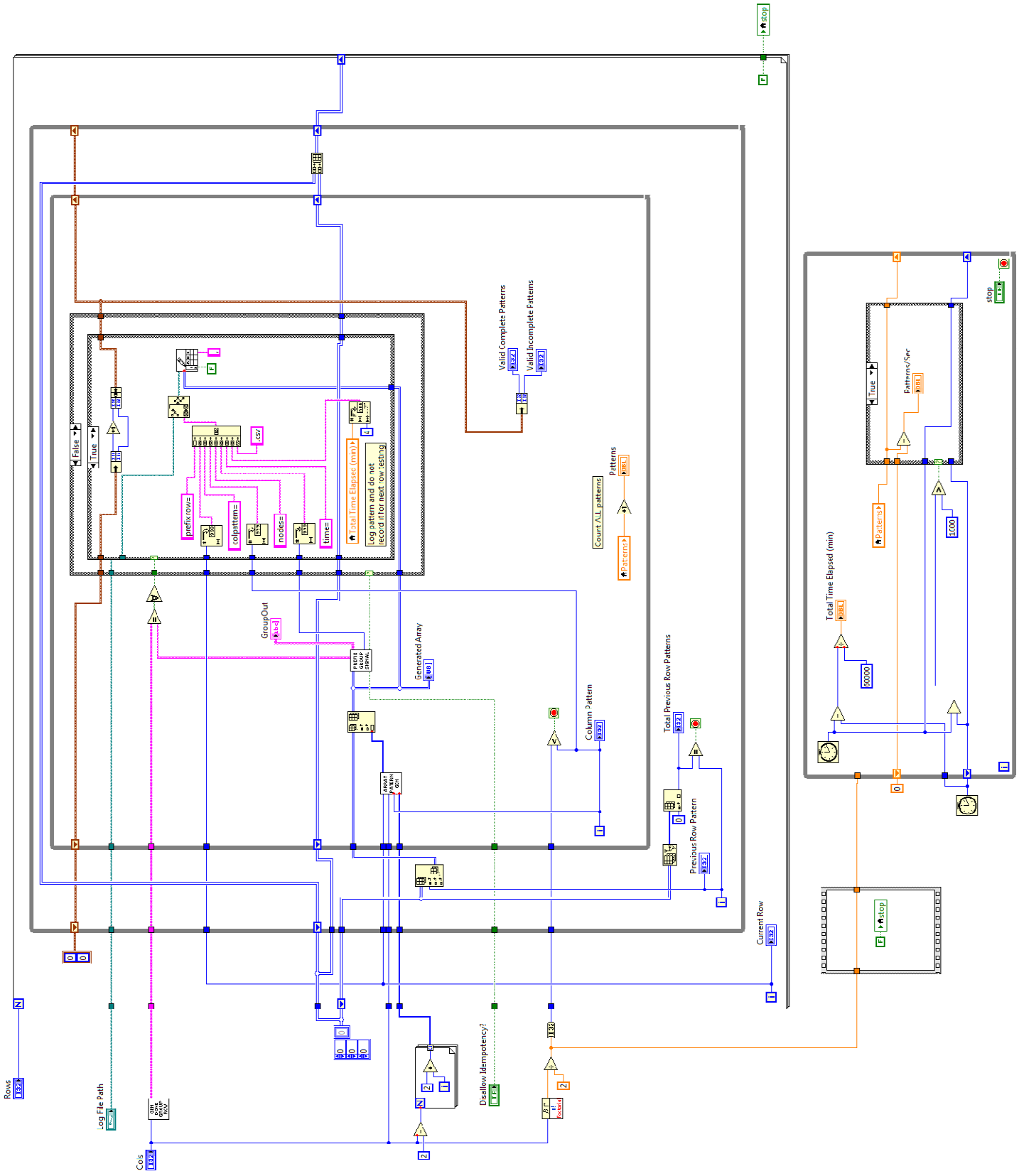
Another possible further extension of this work could be performed by studying the prefix structure results from the search algorithm and potentially come up with some equations that can be used to describe the prefix structure when given key design parameters. This set of equations would be a valuable tool for synthesizers used to translate from RTL to actual transistor or logic implementations in FPGAs or ICs.

6 Appendix A: Search Algorithm LabVIEW Code

6.1 Main Search Algorithm VI Front Panel



6.2 Main Search Algorithm



The **Main Search Algorithm** implements the algorithm described in the Search Algorithm section of this work above.

The algorithm consists of 3 loops:

1. Current Row
2. Previous Row pattern
3. Column pattern

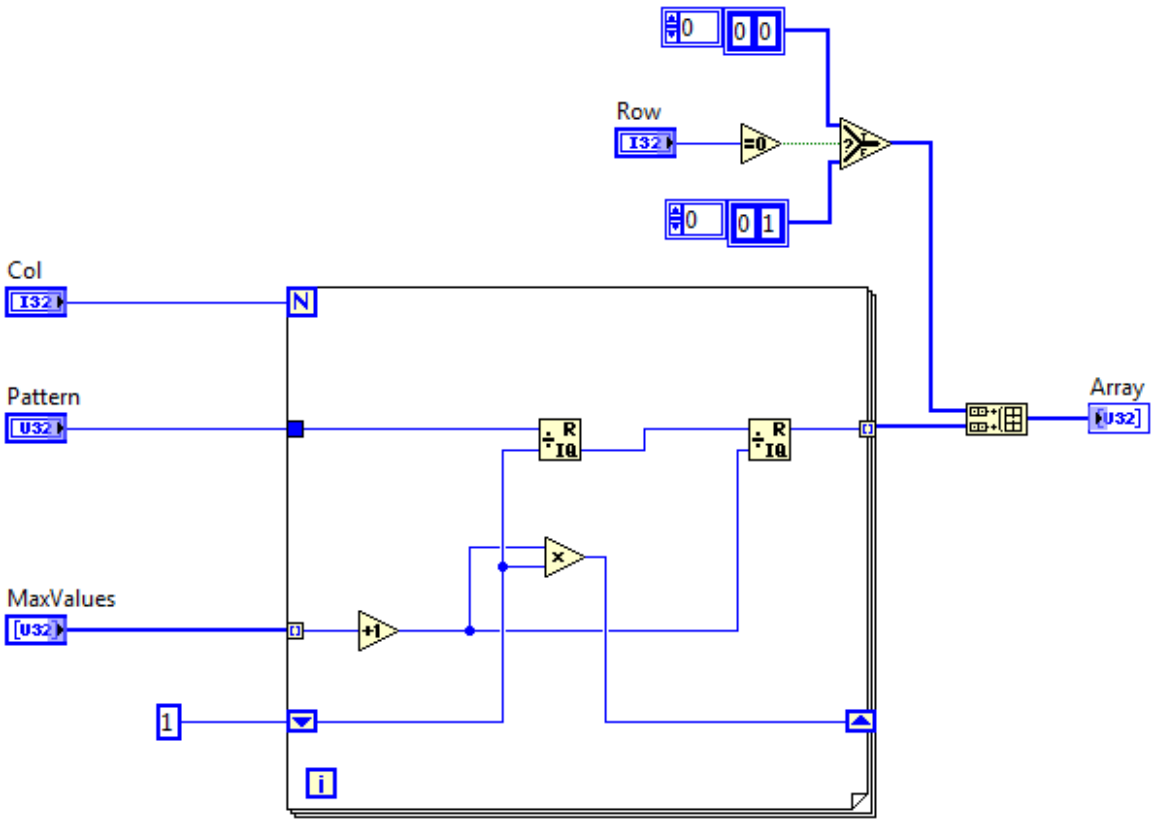
Pseudo-code outline of the algorithm:

1. For the *Current Row*, there will be a collection of valid *Previous Row* patterns to brute force test all *Column* patterns from 0 to (N-1)!-1.
2. If a valid *Column* pattern in the *Current Row* is found and is complete, the structure will be logged as a comma-separated text file for later analysis. Proceed to the next *Column* pattern and repeat steps 2-3.
3. If a valid *Current Row* pattern is found but is incomplete, the structure will be stored in memory as a new *Previous Row* pattern to be used in testing new *Column* patterns in the next *Current Row* iteration. Proceed to the next *Column* pattern and repeat steps 2-3.
4. When all *Column* patterns have been tested, save all the saved incomplete *Current Row* structures and proceed to the next *Previous Row* pattern, repeat steps 2-4.
5. When all the *Previous Row* patterns have been tested, gather all the saved incomplete *Current Row* structures generated from testing all *Column* patterns against all *Previous Row* patterns, and advance to the next *Current Row* using only these incomplete structures for testing new prefix structures. Repeat steps 1-5.

At the completion of this search algorithm, there will be a collection of text files representing valid prefix adder structures that is plotted and analyzed.

The bottom portion of the code is only used to keep track of simulation time and characterize the algorithm pattern test rate on the computer.

6.3 Array Pattern Generator



The Array Pattern Generator subroutine is given a pattern K between 0 to (N-1)!-1 and produces the K^{th} unique N-bit long row pattern. The Row input argument is used to determine if the first 2 bit columns are (0,0) or (0,1) as described by observation #2 in the Search Algorithm section. The MaxValues input is used by the algorithm to determine the maximum value for each row entry as described by observation #1.

Basically, this algorithm does a variable modulo division on the pattern K value so a row R is represented as follows:

R(0) = 0 always

R(1) = 0 (if first row) or 1 (all other rows)

R(2) = K mod 3

R(3) = floor(K/3) mod 4

R(4) = floor(K/12) mod 5

R(5) = floor(K/60) mod 6

R(6) = floor(K/360) mod 7

R(7) = floor(K/2520) mod 8

For example, for N= 8 bits, assuming the first row is being generated:

Pattern K=0 is converted to R={0,0,0,0,0,0,0,0}

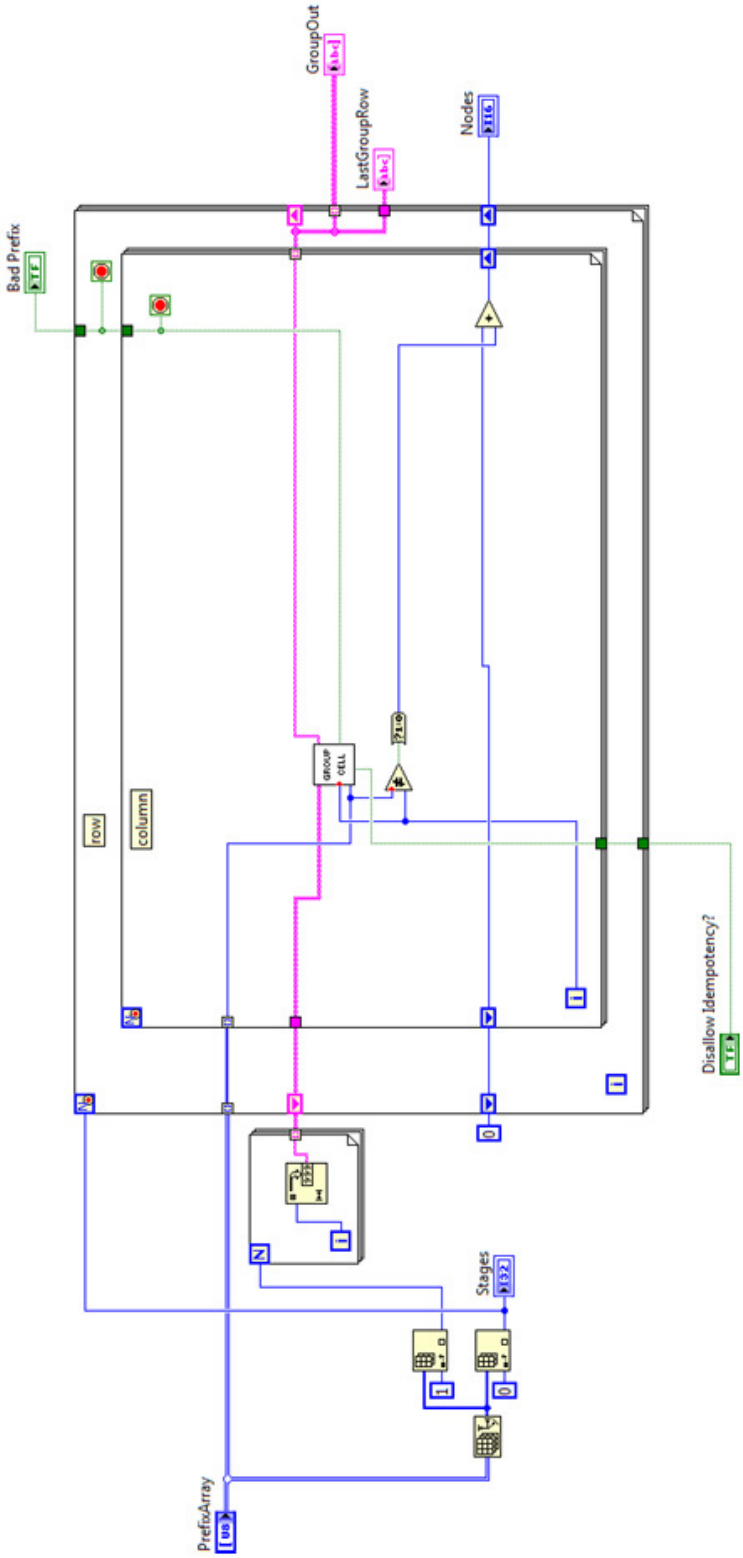
Pattern K=1 is converted to R={0,0,0,0,0,1,0,0}

Pattern K=2 is converted to R={0,0,0,0,0,2,0,0}

Pattern K=3 is converted to R={0,0,0,0,1,0,0,0}

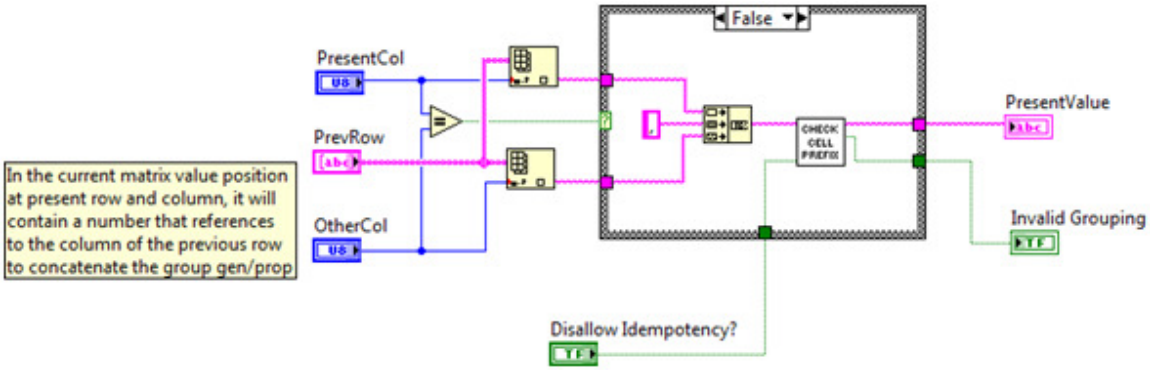
Pattern K=((N-1)!-1)=(7!-1) = 20159 is converted to R={7,6,5,4,3,2,0,0}

6.4 Prefix Group Signals



The **Prefix Group Signals** subroutine is given a prefix array with a minimum of 1 row. A loop iterates through each row and uses the Group Cell subroutine to determine if a row contains a valid pattern, and if so, continue to the next row and repeat the validation. If the **Group Cell** routine detects a bad structure then this subroutine will quit with a Bad Prefix error. If the subroutine completes without error then it will return the total number of nodes used in the prefix structure and the final group generate and propagate signals formed by the prefix structure. The main search algorithm uses the final group generate and propagate signal result to determine if it is complete or incomplete and requires more subsequent row(s) to complete the structure.

6.5 Group Cell

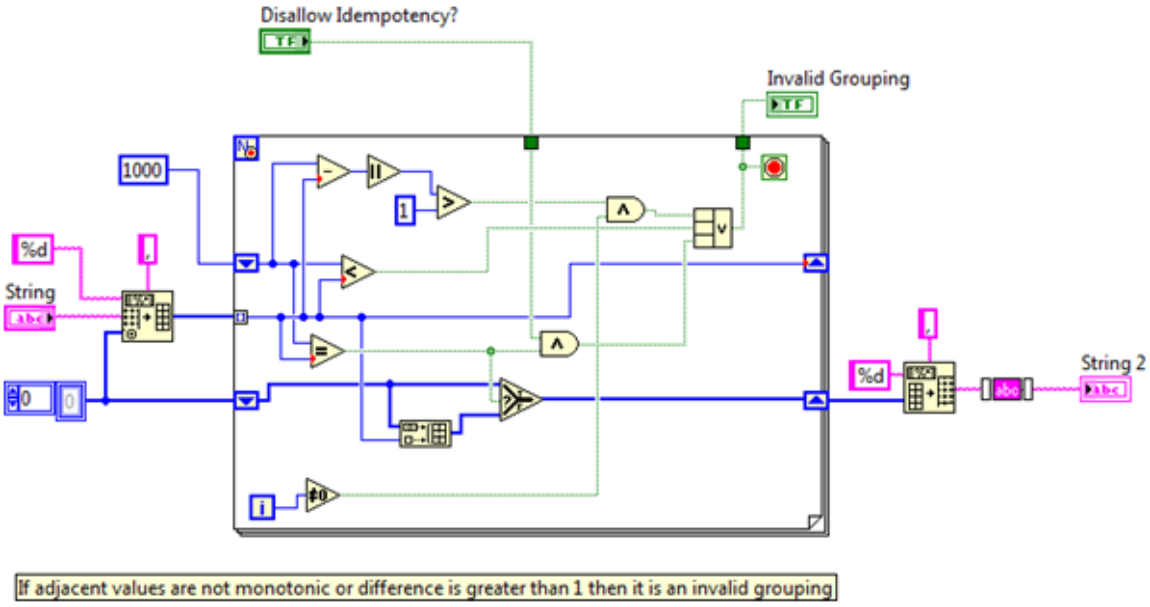


The **Group Cell** subroutine basically acts like a single Node for the i^{th} bit column and verifies that the Node is valid.

The subroutine retrieves the previous row groups (PrevRow array) at the current i^{th} bit column (PresentCol) and at the value contained within the prefix structure for the source group (OtherCol). If the value contained in the prefix structure and bit column i are equal (i.e PresentCol = OtherCol), then a buffer exists rather than a Node and the output group is the same as the original group at the i^{th} bit column position, that is, PresentValue = PrevRow(PresentCol).

The **Check Cell Prefix** subroutine is used to determine if the newly concatenated group signals are valid.

6.6 Check Cell Prefix

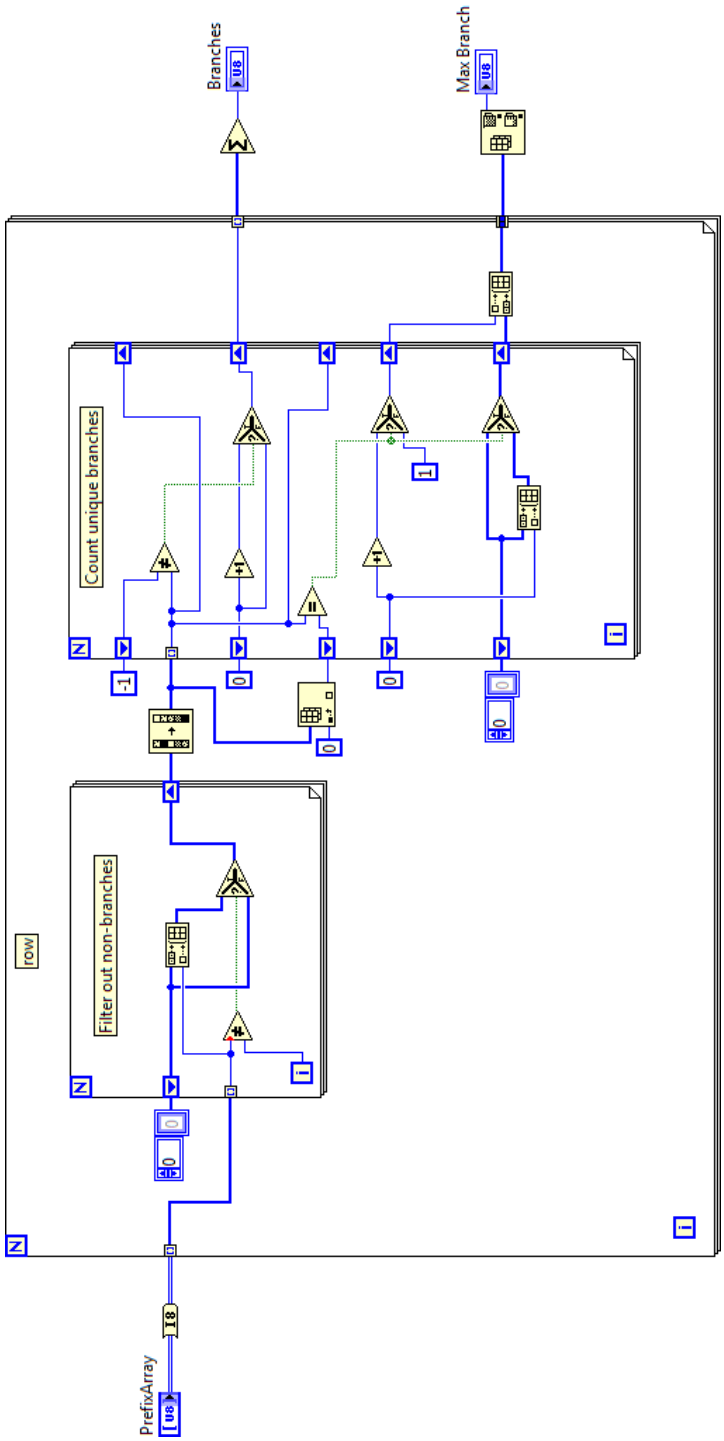


The **Check Cell Prefix** is where the prefix operator rules are applied and tested to verify if a new group generate and propagate signal is valid. The group signals are organized as a comma separated string (String), for example, group signals 3:0 are represented as 3,2,1,0. The prefix test rules are:

1. Groupings may have redundancy if idempotency is allowed by detecting 2 consecutive equal values. For example, grouping 3:2 with 2:1 would be represented as 3,2,2,1 in the String. If idempotency is allowed, then it will eliminate redundancy on the input String and generate a new signal 3:1 represented as 3,2,1 in the output String 2. Otherwise, the grouping is invalid because no redundancy is allowed.
2. Groupings must have monotonic order. For example, 3:0 is represented as 3,2,1,0 and is always monotonic. An invalid grouping of 3:1 and 3:0 is represented as 3,2,1,3,2,1,0, which is not monotonic. This applies both the associativity and idempotency rules, that is, redundancy can only be at most 1 bit and the prefix operator boundary cannot be separated by more than 1 bit.
3. Groupings must not have gaps of more than 1 bit. For example, the string 3,2,0 is invalid because it cannot represent group 3:0 without bit column 1. This is a direct application of the associativity rule.

7 Appendix B: Prefix Structure Analysis LabVIEW Code

7.1 Prefix Branch Count

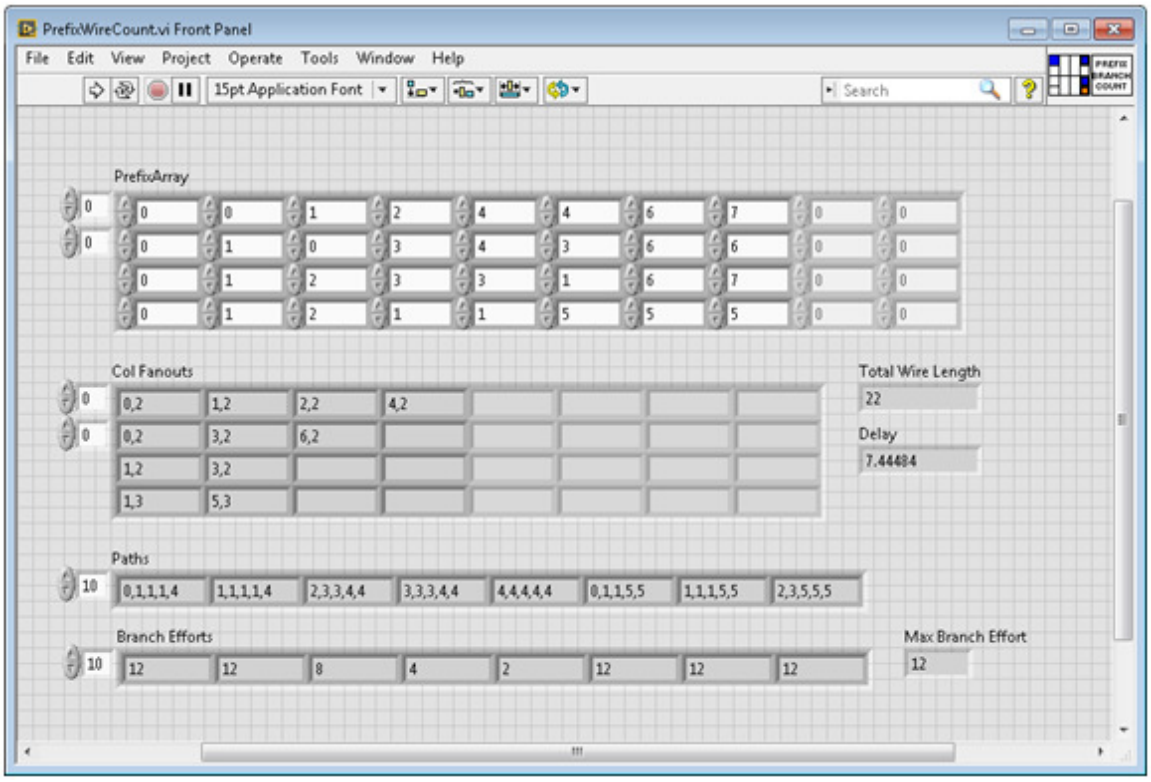


The **Prefix Branch Count** takes in the prefix structure matrix, filters out all columns that do not have a branch, and does two things:

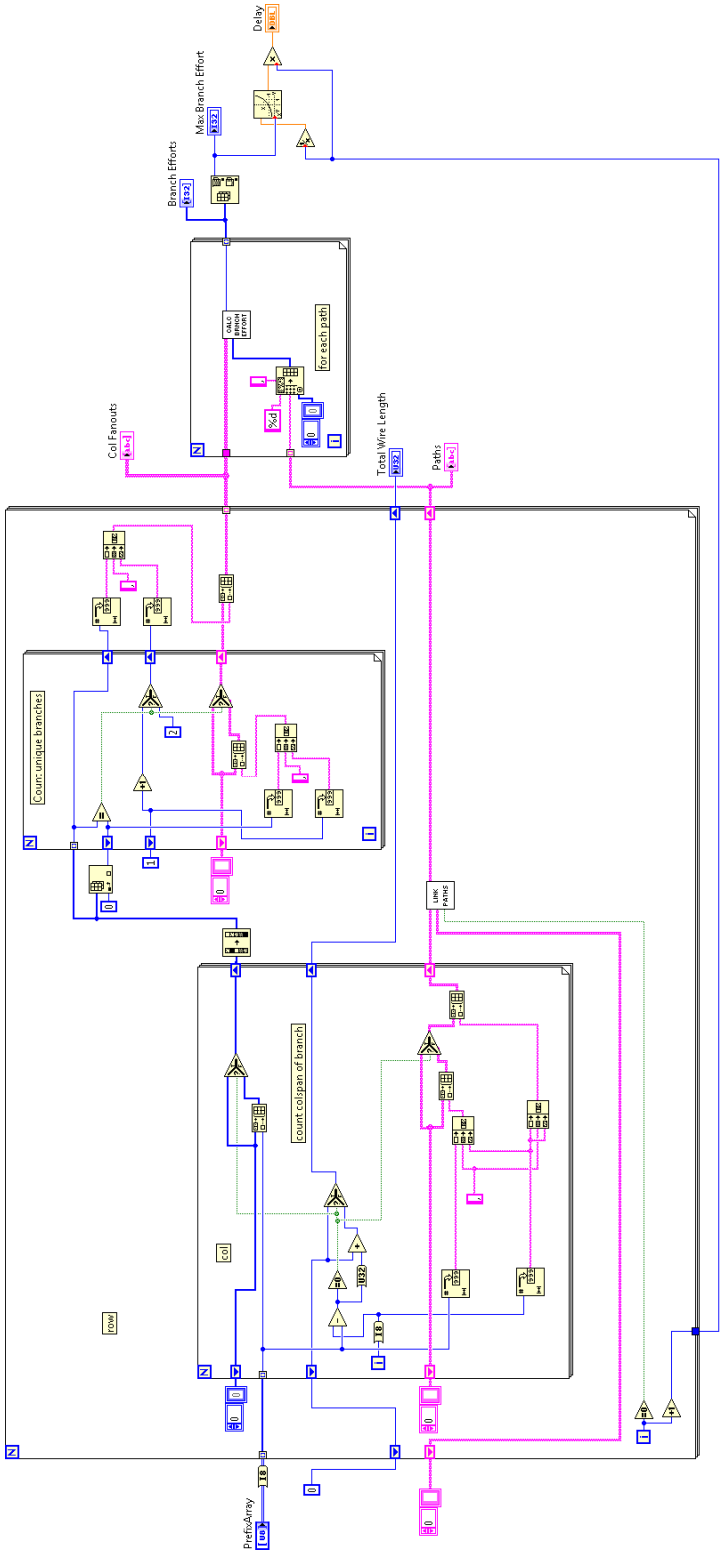
1. Count the total number of branches.
2. Count the total number of branches from the same column and finds the maximum number of branches in the overall design.

This subroutine is used in analyzing prefix adder structures for the Node, Fanout, and Depth study. The nodes and depth come from the file name generated by the search algorithm.

7.2 Prefix Wire Count VI Front Panel



7.3 Prefix Wire Count

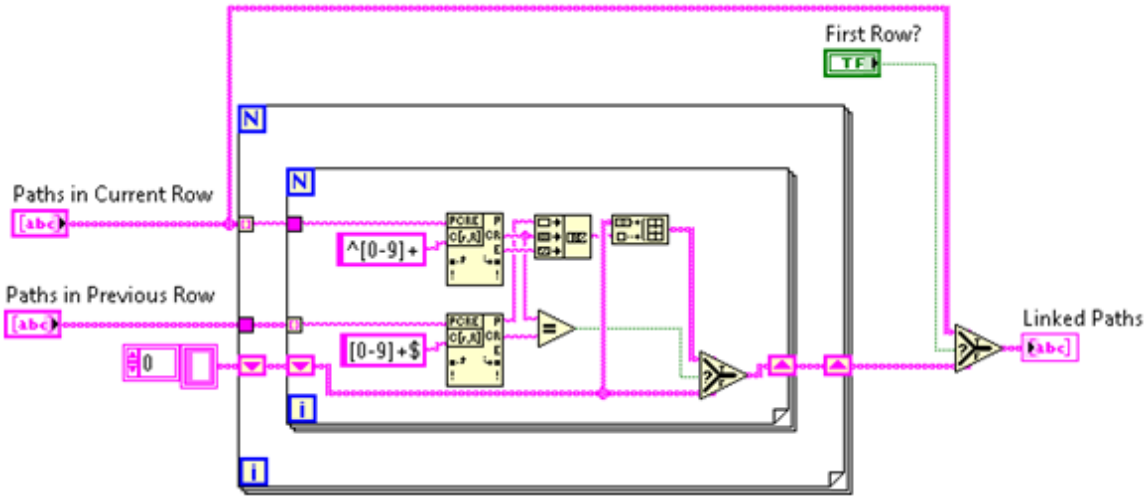


The **Prefix Wire Count** is more involved compared to the Prefix Branch Count and is central to performing the second study approach for calculating the area-power and delay metrics. It takes in the prefix structure matrix and analyzes the paths.

Pseudo-code outline of the subroutine:

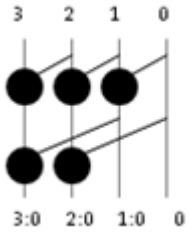
1. For each *Row* in the prefix matrix, iterate through all *Columns* and whenever the value at that location in the matrix is not equal to the *Column* value then it is a branch.
    a. Collect all these branch source columns into an array.
    b. Find the difference between the matrix value and the *Column* value to determine the *Horizontal Wire Length* and accumulate the lengths for the *Row*.
    c. Accumulate a record of paths from the current matrix location at (*Row, Column*). All paths have a branch from (*Row-1, Column*) and potentially another branch from (*Row-1, Matrix Value*) where the *Matrix Value* is the branch Column in the previous row as defined earlier. These paths are stored as a String *source column, final column*.
2. The second inner loop at the top inside the main loop iterates through all the collected branches from step (1a) to generate the fanout count for each *Column* in the current *Row*. If a *Column* doesn't have a fanout of more than 1 then it will not receive a fanout count. The fanout counts are accumulated into a *Col Fanouts* array and each entry is stored as a String *column, fanout*.
3. The **Link Paths** subroutine joins the recorded paths for the *Current Row* with the *Previous Row* and accumulates a growing list of possible paths from the top row down. The paths are stored as a String *source column, intermediate column1, …, intermediate column N, final column* where each entry separated by a comma is for each row. For example, a ripple-carry adder with a diagonal path from bit 0 to 7 would have the following String record 0,1,2,3,4,5,6,7. Also each column would have a path such as 0,0,0,0,0,0,0,0 and so on.
4. After iterating through all *Rows* in the prefix matrix, we now have arrays of records to perform the next step. For all the *Rows*, the accumulated wire length now indicates the *Total Wire Length* for the structure. The next loop iterates through each *Path* record and run the **Calc Branch Efforts** subroutine using the data from the *Col Fanouts* array.
5. Finally, the maximum branch effort is used to calculate the minimum delay.

7.4 Link Paths



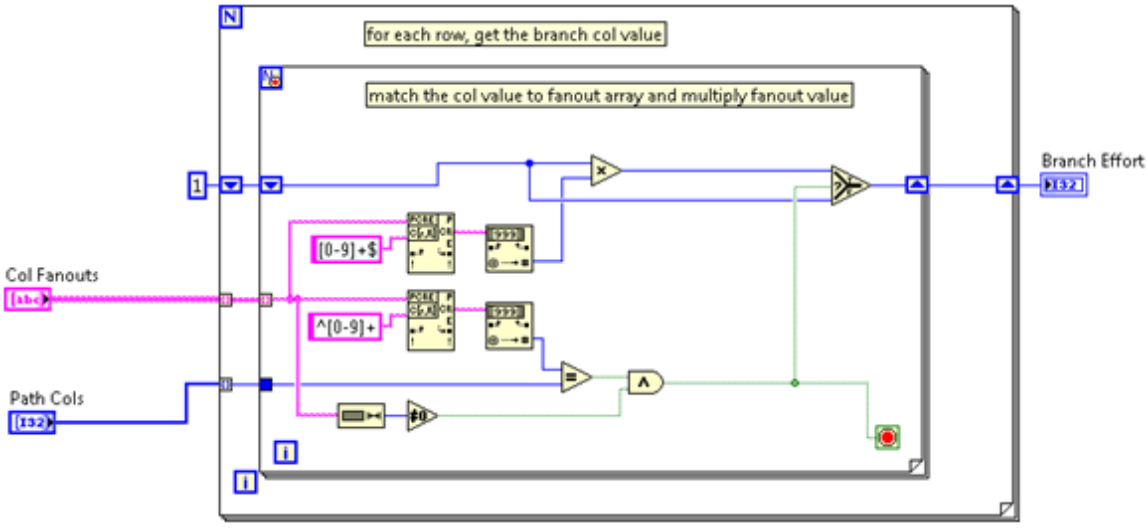
The **Link Paths** subroutine iterates through each column in the current row and then scans through the columns of the previous row to find where the start column value of the current row may match the final column value of the Previous Row and links these paths together into a new concatenated string. In other words, it iterates through the current row to find where the path head matches the tail end of paths from the previous row.

Take the following 4-bit Kogge-Stone as an example:



Since there are only two rows, the first is the previous and the second is the current row. At a minimum, all rows will have vertical paths 0,0 and 1,1 and 2,2 and 3,3. In addition to those, the previous row contains the 3 horizontal branch paths 0,1 and 1,2 and 2,3. The current row contains the 2 horizontal branch paths 0,2 and 1,3. The Link Paths subroutine will link the heads of all current row paths to the tails of all previous row paths, so the following linked paths are created: 0,0,0 and 0,0,2 and 0,1,1 and 1,1,1 and 1,1,3 and 1,2,2 and 2,2,2 and 2,3,3 and 3,3,3.

7.5 Calc Branch Efforts

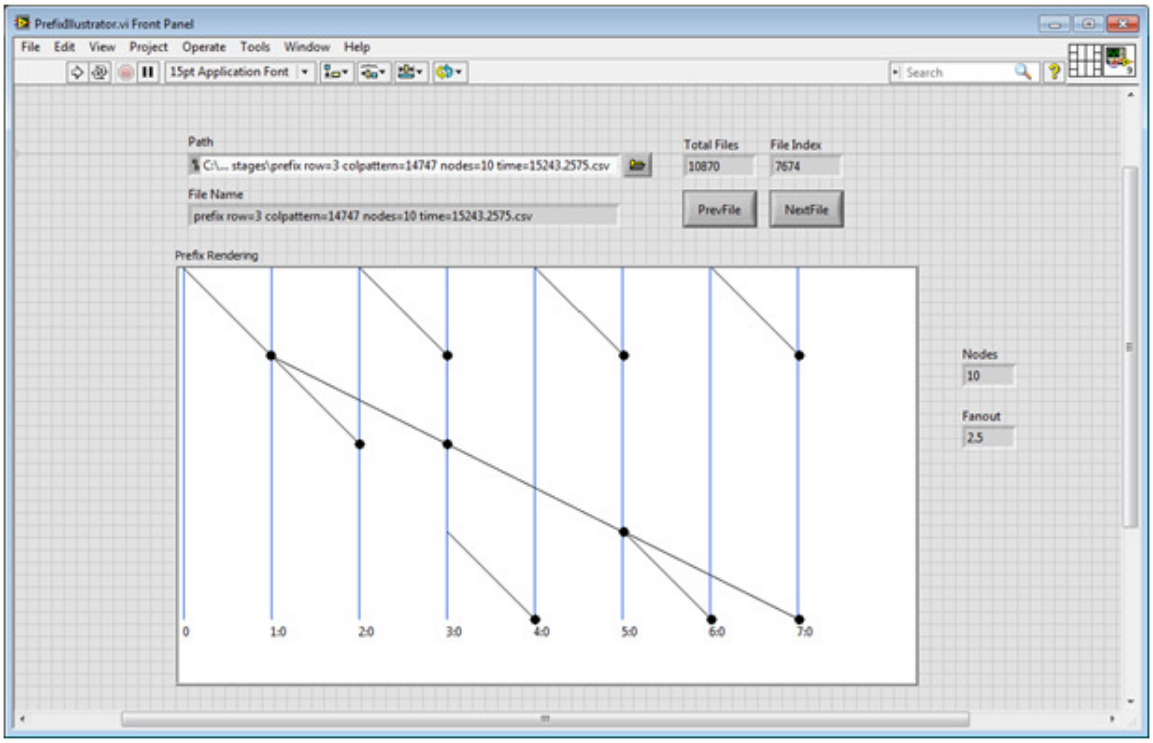


The **Calc Branch Efforts** subroutine takes in the *Path* record which is a String of linked columns from start to end in each row. It iterates through each *Row* and takes that column value from the *Path* record and pulls in the row from the *Col Fanouts* record and determines if the linked column has a fanout value to multiply into the Branch Effort metric. This subroutine is iterated in the **Prefix Wire Count** routine loop to accumulate an array of Branch Efforts for all paths so a worst-case value can be determined.

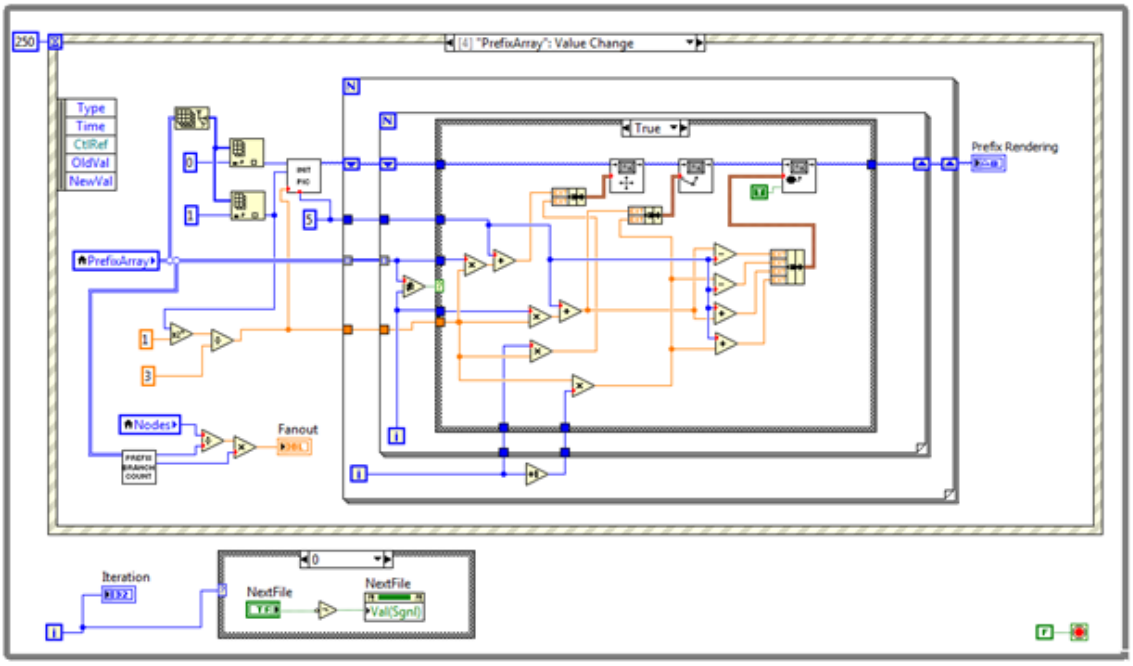
8 Appendix C: Prefix Structure Illustrator LabVIEW Code

8.1 Prefix Structure Illustrator VI Front Panel

This software was written to be able to select any one of the .csv log files containing a valid parallel prefix adder structure produced by the search algorithm software and draw a picture of the structure. This comes in handy for quickly inspecting the log files without having to draw out the structure by hand. This software also helps locate the structures with the desired characteristics (i.e. number of nodes and fanout metric).



8.2 Prefix Structure Illustrator



The Illustrator is written as an Event structure so the Previous/Next file buttons or keyboard strokes can be used to sift through files in a folder and review the prefix structure rendering on the front panel. The above screenshot only shows the key component that prepares and draws the prefix structure image from the prefix array contained within the log files.

One of the subroutines (subVIs) used by this code is the **Prefix Branch Count** and **Prefix Wire Count** to help sift through the results for prefix structures that have appealing characteristics.

No further details will be provided because this code is not central to the search algorithm and is merely a supplementary tool to expedite analysis of the search algorithm results.

9 References

[1] R. Brent and H. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers*, Vol. C-31, 1982, pp. 260-264.

[2] P. Kogge and H. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers*, Vol. C-22, No. 8, 1973, pp. 786-793.

[3] R. Ladner and M. Fischer, "Parallel Prefix Computation," *JACM*, Vol. 27-4, 1980, pp. 831-838.

[4] S. Knowles, "A Family of Adders," *Proc. 15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 277-281.

[5] T. Han and D. Carlson, "Fast Area-Efficient VLSI Adders," *Proc. 8th IEEE Symposium on Computer Arithmetic*, 1987, pp. 49-56.

[6] Y. Choi and E. Swartzlander, "Parallel Prefix Adder Design with Matrix Representation," *IEEE Symposium on Computer Arithmetic*, 2005.

[7] N. Weste and D. Harris, "CMOS VLSI Design", 4th Ed, Addison-Wesley, 2011, pp. 438-456.

[8] D. Harris and I. Sutherland, "Logical Effort of Carry Propagate Adders," *37th Asilomar Conference on Signals, Systems & Computers*, Vol. 1, 2003, pp. 873-878.

[9] T. Callaway and E. Swartzlander, "Estimating the power consumption of CMOS adders," *Proc. 11th IEEE Symposium on Computer Arithmetic*, 1993, pp. 210-216.

VITA

Jonathan Barten Stanley was born in Lewisville, Texas in 1988. After receiving the Bachelor of Science in Electrical Engineering from Rice University, Houston, Texas in May 2010, he started working for National Instruments in Austin, Texas. In January 2011, he entered the Graduate School at The University of Texas at Austin as a part-time student.

Permanent email address: jbstanley@utexas.edu

This report was typed by the author.